Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Marko Struna

Spletni urejevalnik JSON datotek s podano vsebinsko shemo

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Tomaž Dobravec

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom LaTeXbesedil TexMaker.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Zaradi velike prostorske potratnosti in nečitljivosti zapisov v XML formatu se v zadnjem času vedno bolj uveljavlja alternativni način zapisa tekstovnih podatkov v JSON formatu. V diplomskem delu razvijte podatkovno shemo za opis vsebine JSON datoteke. Shema naj predvideva opis podatkov različnih tipov (število, niz, datoteka, tabela, ...). Poiščite obstoječa orodja za urejanje JSON datotek in ugotovite, ali lahko katerega od njih uporabite za urejanje datotek s podano vsebinsko shemo. Razvijte tudi lastno spletno rešitev, ki omogoča urejanje preko vsebinskih shem povezanih datotek. Pri tem uporabite programska jezika Perl in JavaScript ter druge odprtokodne rešitve.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Marko Struna, z vpisno številko **24930355**, sem avtor diplomskega dela z naslovom:

Spletni urejevalnik JSON datotek s podano vsebinsko shemo

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Tomaža Dobravca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 11. junija 2014 Podpis avtorja:

Za pomoč pri izdelavi diplomske naloge se zahvaljujem mentorju doc. dr. Tomažu Dobravcu. Zahvaljujem se tudi družini za spodbudo in razumevanje na moji dolgi poti do diplome.

Svoji dragi ženi in otrokom.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
JSON	JavaScript Object Notation	Opis JavaScript objekta
AJAX	Asynchronous JavaScript and XML	Asinhronini JavaScript in XML
YAML	YAML Ain't Markup Language	YAML ni opisovalni jezik
DOM	Document Object Model	Objektni model dokumenta

Povzetek

V diplomski nalogi je predstavljena rešitev za sistematično urejanje JSON datotek preko spletnega vmesnika.
Predstavljena rešitev omogoča enostavno urejanje JSON datotek preko podane vsebinske sheme, kjer so opisane lastnosti vseh elementov. Shema vsebuje elemente različnih podatkovnih tipov, kjer imajo nekateri opcijske parametre kot so npr. validacija vhodnega niza, minimalno število,... V shemi je tudi napisano mesto vključene datoteke.
Predstavljena rešitev je bila implementirana kot spletna stran. Izdelana je bila delno v programskih jezikih Perl, Javascript, delno pa v jQuery. Za opis spletnih datotek se uporablja HTML::Template system predlog, za urejanje besedil in implementacijo urejevalnika datotek smo si pomagali z odprtokodnimi rešitvami.
Prednost v primerjavi z že obstoječami rešitvami za urejanje JSON datotek je predvsem ta, da je tu možno urejati več JSON datotek hkrati, ki so povezane preko vsebinskih shem.

Ključne besede: JSON datoteke, spletni vmesnik, predloge, HTML, AJAX, JavaScript, jQuery, Perl.

Abstract

The thesis presents a solution for editing JSON files using a web interface. The presented solution allows editing of JSON files using a content schema, where properties of all elements for the JSON file are described. The schema contains elements of different data types. Each of them can be acompanied by optional parameters (i.e. such parameters are text validation, minimum allowed number,...).
The solution was implemented as a Web application. *Perl*, *JavaScript* and *jQuery* tehnologies were used. To describe a *HTML* file the `HTML::Templates` system is used, for text editing and the implementation of the file manager existing opensource solutions were used.
Advantage of implemented solution over existing solutions for editing JSON files is that it allows editing of multiple connected JSON files simultaneously on one web page.

Keywords: JSON files, web server, templates, HTML, AJAX, JavaScript, jQuery, Perl.

Poglavje 1

Uvod

1.1 Namen dela

Urejanje JSON datotek zna biti časovno zahtevno in tudi zelo hitro lahko pride do kakih napak. Namen diplomskega dela je poenostaviti urejanje skupine JSON datotek, ki se uporabljajo pri določenem projektu, ne da bilo treba pri tem še paziti na datotečno strukturo.
Vsaka JSON datoteka ima glede na tip entitete določeno vsebinsko shemo, ki opisuje to entiteto. Shema vsebuje ime entitete, pot do datoteke, vrstni red elementov na spletni strani ter opis lastnosti.
Program zelo olajša delo pri izdelovanju in urejanju projektov. Ni potrebno skrbeti v katero mapo se bo kaj shranilo, poleg tega vemo, da bo vsebina vseh JSON datotek vedno pravilna.

1.2 Zgradba dela

V prvem poglavju je na splošno predstavljeno diplomsko delo, namen in njegova zgradba.
Drugo poglavje predstavlja tehnologijo, ki je bila uporabljena za izdelavo diplomskega dela.
Sledi poglavje, kjer so predstavljene uporabljene odprtokodne rešitve za ure-

janje vsebine datotek.

V četrtem poglavju je opisana implementacija rešitve.

Sledi zaključek.

V dodatku so pripete sheme in generirane datoteke iz primera.

Poglavje 2

Tehnologije

2.1 JSON datoteke

JSON (JavaScript Object Notation) je odprtokodni standard tekstovnih datotek za izmenjavo podatkov v človeku razumljivi obliki in je poleg tega enostaven za strojno razčlenjevanje in generiranje [6]. Definiran je kot podmnožica programskega jezika JavaScript. JSON je tekstovni format, ki je neodvisen od programskega jezika, a uporablja dogovore, ki so poznani programerjem v jezikih, podobnim jeziku „C“, vključno s C, C++, C#, Java, JavaScript, Perl, Python in mnogi drugi. Zaradi teh lastnosti je JSON popoln za prenos podatkov.
JSON je zgrajen iz dveh struktur:

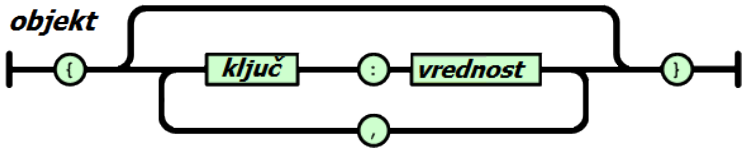
- Zbirka parov ključ/vrednost. V različnih jezikih je to realizirano kot objekt, zapis, struktura, slovar, asociativna tabela, povezan seznam ali asociativni seznam.

- Urejen seznam vrednosti. V večini jezikov je to predstavljeno kot tabela, vektor, seznam ali zaporedje.

To sta splošno namenski podatkovni strukturi. Praktično vsi moderni programski jeziki ju podpirajo v eni ali drugi obliki. Smiselno je, da je format

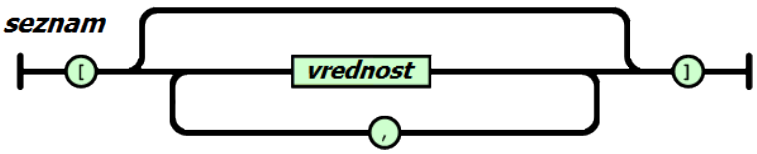
podatkov, ki je prenosljiv med programskimi jeziki, tudi izpeljan iz teh struktur.

Objekt je neurejen par ključev/vrednosti. Objekt se prične z "{"(zavit oklepaj) in se konča z "}"(zaviti zaklepaj). Vsakemu imenu sledi znak ":"(dvopičje), pari ključ/vrednost pa so ločeni z ","(vejico).



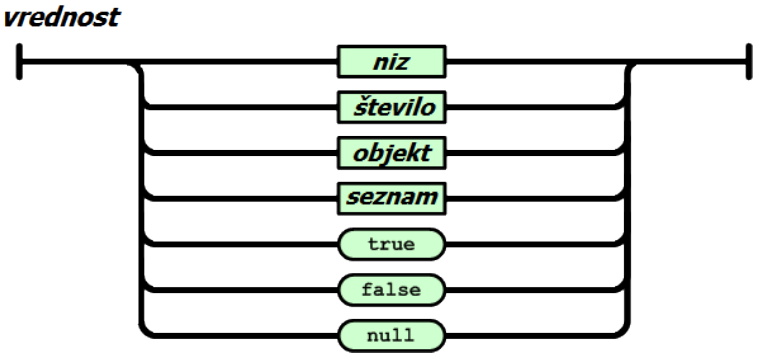
Slika 2.1: Deklaracija objekta

Array (seznam) je urejena zbirka vrednosti. Seznam se prične z "["(oglati oklepaj) in se konča z "]"(oglati zaklepaj). Vrednosti so med seboj ločene z ","(vejico).



Slika 2.2: Deklaracija seznama

Value (vrednost) je lahko niz med dvojnimi narekovaji, število, `true` ali `false` ali `null`, objekt ali seznam. Te strukture so lahko vgnezdene.

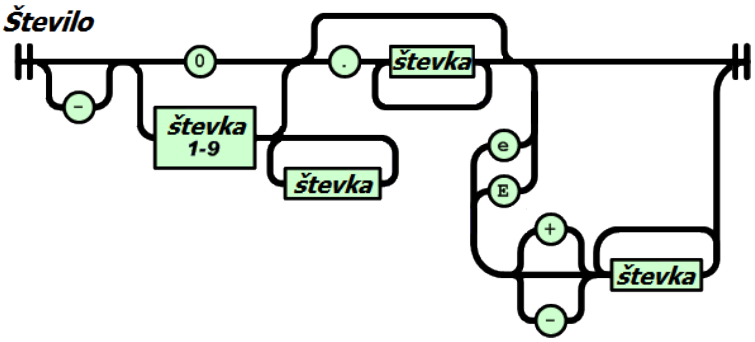


Slika 2.3: Deklaracija vrednosti

String (niz) je zaporedje nič ali več Unicode znakov, vpetih med dvojne narekovaje, uporabljajo pa se lahko tudi t.i. escape kode. Znak predstavlja niz dolžine 1. Niz je zelo podoben C ali Java nizu.

Slika 2.4: Deklaracija niza

Number (število) je zelo podobna številu v C ali Java, le da se osmiški in šestnajstiški zapis ne uporabljata.



Slika 2.5: Deklaracija števila

Takoimenovani beli znaki so lahko vrinjeni kamorkoli med posebne znake. Z izjemo nekaterih kodirnih detajlov je tako jezik popolnoma opisan.

Primer JSON datoteke:

```
{
   "TestSet" : {
      "TestRepeat" : "10",
      "ShortName" : "Test2",
      "N" : "45",
      "TestSetFiles" : [
         "testset2.txt"
      ],
      "Desc" : "Test RND, INVERSED and SORTED by increasing n",
      "Name" : "Test2"
   }
}
```

2.2 Ajax

Kratica Ajax pomeni Asynchronous JavaScript and XML [8]. Ajax sodi med novejše tehnike za ustvarjanje boljših, hitrejših in bolj interaktivnih spletnih strani, pri čemer pomaga XML, HTML, CSS in JavaScript.

Ajax uporablja XHTML za vsebino in CSS za njeno predstavitev, za dinamični prikaz se uporabljata tudi Document Object Model in JavaScript. Pri običajnih spletnih straneh poteka prenos informacij do strežnika in nazaj s pomočjo zaporednih spletnih zahtev. To pomeni, da uporabnik najprej izpolni formo, pritisne gumb Submit, nakar se trenutna spletna stran zamenja s stranjo, ki vključuje nove informacije, pridobljene s strežnika. Pri Ajaxu se pošljejo podatki na spletni strežnik s pomočjo JavaScripta, ki nato nove informacije obdela in posodobi trenutno spletno stran. Za prenos podatkov se običajno uporabi XML, lahko pa se uporabi kakršenkoli format, tudi običajni tekst.
Ajax je spletna tehnologija neodvisna od programske opreme spletnega strežnika. Uporabnik lahko naprej uporablja spletno aplikacijo, medtem ko se podatki prenašajo do strežnika in nazaj v ozadju. Za izvajanje Ajax je dovolj katerikoli dogodek na spletni strani, ki sproži izvajanje funkcije.

Ajax temelji na naslednjih odprtih standardih

HTML in CSS

Za prikaz spletne strani je dovolj uporaba HTML tehnologije za prikaz vsebina, za urejenost in postavitev strani skrbi CSS

XML format

Običajni format podatkov, ki se prenaša s strežnika k uporabniku, je zapisan v XML formatu. Možno je seveda uporabiti tudi tekst v normalni obliki, vendar se XML uporablja pogosteje.

XMLHttpRequest objekt

Za komunikacijo med stranjo na uporabniški strani in spletnim strežnikom skrbi JavaScript objekt XMLHttpRequest

JavaScript

Za samo delovanje Ajax tehnologije je zelo pomemben JavaScript programski jezik, kjer je definiran objek XMLHttpRequest preko katerega

se naredi zahtevek na spletno stran in ki vrne podatke s strežnika k uporabniku, kjer se pridobljene informacije obdelajo in z njihovo pomočjo se nato stran posodobi.

2.2.1 Tehnologije, ki so v uporabi za Ajax

JavaScript

To je programski jezik, kjer se funkcije izvedejo, ko se sproži določen dogodek. JavaScript povezuje celotno Ajax tehnologijo.

DOM

DOM predstavlja API funkcije za dostop in upravljanje sktruturiranih dokumentov. DOM predstavlja strukturo XML in HTML dokumentov.

CSS

CSS omogoča popolno ločevanje predstavitve od vsebine. Predstavitveni stil se lahko tudi programsko spreminja s pomočjo JavaScripta.

XMLHttpRequest

To je JavaScript objekt, ki omogoča asinhrono interakcijo s spletnim strežnikom.

2.2.2 Primeri uporabe Ajax

Google Maps Uporabnik lahko premika zemljevid ne da bi bilo potrebno pritisniti na katerikoli gumb.
http://maps.google.com/

Google suggest Med tipkanjem Google ponuja najrazličnejše predloge, s smernimi tipkami je možno izbrati eno izmed možnosti.
http://www.google.com/webhp?complete=1&hl=en

Gmail Spletna pošta
http://www.gmail.com/

2.2.3 Podpora brskalnikov za Ajax

Vsi brskalniki, ki so na voljo, ne podpirajo Ajax. Tukaj je podan seznam najbolj pogostih brskalnikov, ki podpirajo Ajax.

- Mozilla Firefox 1.0 in višje
- Netscape version 7.1 in višje
- Google Chrome
- Apple Safari 1.2 in višje
- Microsoft Internet Exporer 5 in višje
- Konqueror
- Opera 7.6 in višje

Opomba Ko rečemo, da brskalnik ne podpira Ajax, to preprosto pomeni le to, da ni podprt JavaScript objekt XMLHttpRequest.

Pisanje kode prenosljive med brskalniki

Preprost način za pisanje izvorne kode, ki je kompatibilna z večino brskalnikov, je uporaba `try...catch` blokov v JavaScript kodi.

```
<html>
    <body>
        <script language="javascript" type="text/javascript">
        <!--
        // Koda, ki podpira vec brskalnikov
        function ajaxFunction() {
            var ajaxRequest;  // Spremenljivka, ki omogoca Ajax!

            try {
                // Opera 8.0+, Firefox, Chrome, Safari
                ajaxRequest = new XMLHttpRequest();
            } catch (e) {
```

```
                // Brskalniki Internet Explorer
                try {
                    ajaxRequest = new ActiveXObject("Msxml2.
                       XMLHTTP");
                } catch (e) {
                    try {
                        ajaxRequest = new ActiveXObject("
                           Microsoft.XMLHTTP");
                    } catch (e) {
                        // Nekaj je slo narobe
                        alert("Brskalnik je pocil!");
                        return false;
                    }
                }
            }
        }
        //-->
        </script>
        <form name='mojaForma'>
            Ime: <input type='text' name='uporabnik' /> <br />
            Cas: <input type='text' name='cas' />
        </form>
    </body>
</html>
```

V zgoraj podani JavaScript kodi se poskusi na tri načine narediti XMLHttpRequest objekt.

Prvi poskus je za Opera 8.0+, Firefox, Chrome in Safari:

```
ajaxRequest = new XMLHttpRequest();
```

Če je to neuspešno, se poskuša narediti pravi objekt za Internet Explorer z ukazoma:

```
ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
```

Če niti to ni uspešno, to pomeni, da je v uporabi zelo zastarel brskalnik, ki ne podpira XMLHttpRequest, kar pomeni tudi to, da ne podpira Ajax.

Po vsej verjetnosti je sedaj spremenljivka `ajaxRequest` nastavljena na tisti *XMLHttpRequest* standard, ki ga trenutni brskalnik uporablja, in pošiljanje podatkov na strežnik se lahko prične.

2.2.4 Ajax v akciji

V tem delu je predstavljen proces izvajanja pri Ajax.

Koraki pri Ajax operaciji

- Na klient strani se sproži nek dogodek.
- XMLHttpRequest objekt je ustvarjen.
- XMLHttpRequest objekt je nastavljen na prave vrednosti
- XMLHttpRequest objekt sproži asinhrono zahtevo na spletni strežnik
- Spletni strežnik vrne rezultat, ki vsebuje dokument v XML obliki
- XMLHttpRequest objekt pokliče *povratno (callback)* funkcijo, ki procesira rezultate
- HTML DOM je posodobljen

Sproži se dogodek

Na uporabniški strani mora priti do nekega dogodka. To je lahko premik miške, pritisk na gumb na miški, pritisk neke tipke na tipkovnici,... Ob določenem dogodku se izvede neka JavaScript funkcija, ki sproži Ajax klic na strežniku.
Primer: Funkcija *validateUserId()* je povezana kot upravljalnik dogodka za *onkeyup* pri vhodnem polju forme, ki ima id nastavljen na *"userid"*

```
<input type="text"  size="20" id="userid" name="id" onkeyup="
   validateUserId();"> }
```

XMLHttpRequest objekt je ustvarjen

```
var ajaxRequest;  // Spremenljivka, ki omogoca Ajax!

function ajaxFunction() {
    try {
        // Opera 8.0+, Firefox, Chrome, Safari
        ajaxRequest = new XMLHttpRequest();
    } catch (e) {
        // Brskalniki Internet Explorer
        try {
            ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
        } catch (e) {
            try {
                ajaxRequest = new ActiveXObject("Microsoft.
                    XMLHTTP");
            } catch (e) {
                // Nekaj je slo narobe
                alert("Brskalnik je pocil!");

                return false;
            }
        }
    }
}
```

XMLHttpRequest objekt je nastavljen na prave vrednosti

V tem koraku je napisana funkcija, ki se bo pognala ob nekem dogodku. Tu je tudi registrirana povratna funkcija funkcija `processRequest()`.

```
function validateUserId() {

   ajaxFunction();

   // Tu je processRequest() povratna funkcija.
   ajaxRequest.onreadystatechange = processRequest;

   if (!target) target = document.getElementById("userid");
```

```
   var url = "validate?id=" + escape(target.value);

   ajaxRequest.open("GET", url, true);
   ajaxRequest.send(null);
}
```

XMLHttpRequest objekt sproži asinhrono zahtevo na spletni strežnik

Izvorna koda je na voljo v prejšnjem primeru. Koda, napisana med nizoma znakov "//————————", je odgovorna za sprožitev zahteve na spletnem strežniku. Vse delo opravi XMLHttpRequest objekt ajaxRequest.

```
function validateUserId() {

   ajaxFunction();

   // Tu je processRequest() povratna funkcija.
   ajaxRequest.onreadystatechange = processRequest;

// --------------------
   if (!target) target = document.getElementById("userid");
   var url = "validate?id=" + escape(target.value);

   ajaxRequest.open("GET", url, true);
   ajaxRequest.send(null);
// --------------------
}
```

Če uporabnik napiše Miha v vhodno polje, potem se v zgornji zahtevi spletni naslov nastavi na `validate?id=Miha` .

Spletni strežnik vrne rezultat, ki vsebuje dokument v XML obliki

Na strežniku je lahko skripta napisana v kateremkoli jeziku. Važno je le, da je logika implementirana na sledeči način.

- Pridobi zahtevo s strani klienta

- Razčleni podatke, ki jih je poslal klient
- Procesiraj podatke
- Pošlji odgovor klientu

Če bi bil v uporabi servlet, bi bil lahko napisan na naslednji način.

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                      throws IOException, ServletException
{
   String targetId = request.getParameter("id");

   if ((targetId != null) &&
       !accounts.containsKey(targetId.trim()))
   {
      response.setContentType("text/xml");
      response.setHeader("Cache-Control", "no-cache");
      response.getWriter().write("true");
   }
   else
   {
      response.setContentType("text/xml");
      response.setHeader("Cache-Control", "no-cache");
      response.getWriter().write("false");
   }
}
```

XMLHttpRequest objekt pokliče *povratno* funkcijo, ki procesira rezultate

XMLHttpRequest objekt je bil nastavljen tako, da pokliče `processRequest()` funkcijo, kjer se stanje XMLHttpRequest objekta nastavi na `readyState`. Sedaj bo ta funkcija sprejela rezultat s strežnika in bo opravila vse potrebno procesiranje. Spodnji primer nastavi vsebino spremenljivke message glede na vsebino podatkov, prejetih s strežnika.

```
function processRequest() {
   if (req.readyState == 4) {
      if (req.status == 200) {
         var message = ...; // nastavi se prava vrednost
         ...
         setMessageUsingDOM(message); // klic funkcije, ki bo
            prikazana v naslednjem poglavju
         ...
      }
   }
}
```

HTML DOM je posodobljen

To je zadnji korak in tu se posodobi HTML stran. To se zgodi v naslednjem vrstem redu.

- JavaScript lahko dobi sklic na katerikoli element na spletni stran z uporabo DOM API.
- Priporočljiv način za pridobitev licence je naslednji način

  ```
  document.getElementById("sporocilo"),
  // kjer je "sporocilo" id elementa,
  // ki se nahaja na HTML strani
  ```

- JavaScript tehnologija se lahko sedaj uporabi za spreminjanje atributov elementa, spreminjanje stila elementa, ali pa za dodajanje, odvzemanje in spreminjanje podelementov. Tu je primer funkcije, ki jo kliče funkcija `processRequest`, ki je predstavljena v prejšnjem poglavju.

  ```
  <script type="text/javascript">
  <!--
   function setMessageUsingDOM(message) {
      var userMessageElement =
             document.getElementById("sporocilo");
      var messageText;
      if (message == "false") {
  ```

```
            userMessageElement.style.color = "red";
            messageText = "Invalid User Id";
        } else {
            userMessageElement.style.color = "green";
            messageText = "Valid User Id";
        }
        var messageBody = document.createTextNode(messageText);

        // ce je messageBody element kreiran, ga preprosto
        // zamenjaj, sicer dodaj novi element
        if (userMessageElement.childNodes[0]) {
            userMessageElement.replaceChild(messageBody,
            userMessageElement.childNodes[0]);
        } else {
            userMessageElement.appendChild(messageBody);
        }
    }
    -->
    </script>
    <body>
    <div id="userIdMessage"><div>
    </body>
```

2.3 JavaScript

JavaScript je programski jezik [9]. Razvit je bil pri Sun Microsystems z namenom, da bi razvili metodo, s kateri bi bilo možno upravljati spletno stran na uporabnikovem računalniku ne da bi bilo potrebno osvežiti spletno stran. Prvi brskalnik, ki je podpiral JavaScript, je bil Netscape 2.0B3. Internet Explorer nikoli ni podpiral pravi JavaScript ampak je imel vgrajeno podporo za Microsoftovo verzijo JavaScript, imenovano JScript, ki je 99% kompatibilen z JavaScript.

Vse moderne spletne strani uporabljajo JavaScript [10]. JavaScript programski jezik je dokaj enostaven za učenje.

2.3.1 Zakaj je potrebno znanje JavaScript?

JavaScript je eden izmed treh jezikov, ki naj bi jih poznali vsi spletni razvijalci:

- **HTML** za definicijo vsebine spletne strani
- **CSS** za določanje izgleda spletne strani
- **JavaScript** za programiranje obnašanja spletne strani

2.3.2 JavaScript lahko spreminja HTML elemente

HTML DOM (the Document Object Model) je uradni W3C standard za dostop do HTML elementov. JavaScript lahko manipulira DOM (spreminja vsebino HTML).
Sledeči primer spremeni vsebino (innerHTML) HTML elementa z id=”demo”.

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

Metoda **document.getElementById()** je ena izmed mnogih metod, deklariranih v HTML DOM.
JavaScript se lahko uporabi za:

- spreminjanje HTML elementov
- brisanje HTML elementov
- kreiranje HTML elementov
- kopiranje in kloniranje HTML elementov
- ... in še veliko več

2.3.3 Bolj konkretni primer, kaj zmore JavaScript

- spreminjanje atributov HTML elementov

```
<!DOCTYPE html>
<html>
<body>
<script>
function spremeniSliko() {
    var image = document.getElementById('myImage');
    if (image.src.match("bulbon")) {
        image.src = "pic_bulboff.gif";
    } else {
        image.src = "pic_bulbon.gif";
    }
}
</script>

<img id="myImage" onclick="spremeniSliko()" src="pic_bulboff
   .gif" width="100" height="180">

<p>Pritisnite na zarnico.</p>

</body>
</html>
```

JavaScript lahko spremeni skoraj katerikoli atribut HTML elementa.

- spreminjanje HTML stila (CSS) Spreminjanje CSS je ena varianta spreminjanja atributov.

```
document.getElementById("demo").style.fontSize = "25px";
```

JavaScript lahko spremeni skoraj katerokoli CSS vrednost.

- validacija vhodnih podatkov

```
<!DOCTYPE html>
<html>
<body>

<p>Vnesite stevilo:</p>

<input id="numb" type="text">
```

```
<button type="button" onclick="myFunction()">Klikni me!</
    button>

<p id="demo"></p>

<script>
function myFunction() {
    // Pridobi vrednost vhodnega polja z id="numb"
    var val = document.getElementById("numb").value;

    // Pridobi elemnt z id="demo"
    var elem = document.getElementById("demo");

    // Ce je vrednost prazna ali ni stevilo
    if ((val.trim() == "") || isNaN(val)) {
        elem.innerHTML = "Ni stevilo.";
    } else {
        elem.innerHTML = "Vhodna vrednost je v redu.";
    }
}
</script>

</body>
</html>
```

2.3.4 Kje se JavaScript koda lahko nahaja?

JavaScript koda se nahaja med elementoma

`<script>...</script>`

Koda je lahko del spletne strani na več načinov:

- Znotraj `<HEAD>...</HEAD>`

```
...
<head>
    <script>
```

```
    function myFunction() {
        document.getElementById("demo").innerHTML = "
            Odstavek spremenjen.";
    }
    </script>
</head>
...
```

- Znotraj <BODY>...</BODY>

```
...
<body>
    <h1>Moja spletna stran</h1>

    <p id="demo">Odstavek</p>

    <button type="button" onclick="myFunction()">Poskusi</
        button>

    <script>
        function myFunction() {
            document.getElementById("demo").innerHTML = "
                Odstavek  spremenjen.";
        }
    </script>
</body>
...
```

- Dodano kot referenca na zunanjo datoteko

```
<!DOCTYPE html>
<html>
    <body>
        <script src="myScript.js"></script>
    </body>
</html>
```

2.4 Perl

Perl je kratica za "Practical Extraction and Report Language"[22]. Perl je skriptni jezik, ki uporablja podobno sintakso kot C/C++. Pogosto ga uporabljajo spletni razvijalci za razvoj skript, ki tečejo na spletnem strežniku. Perl je še posebej uporaben pri razčlenjevanju teksta, zato ga programerji pogosto uporabljajo za branje in iskanje v tekstovnih datotekah.
Jezik je nastal leta 1987, avtor je Larry Wall [23]. Iz Unix programskega jezika se je Perl razvil v močno interentno orodje. Zasnovan je bil kot orodje za hitre popravke v Unix sistemu. Več o jeziku je napisano v [4].
Običajno uporabniki ne bodo videli Perl jezika v akciji, saj se večina dela opravi v ozadju. Perl skripte tečejo na spletnem strežniku še preden se informacije prenesejo v brskalnik.

2.4.1 Perl končnica

Perl skripta se lahko naredi v kateremkoli tekstovnem editorju. Ne glede na to, katero orodje se uporablja za urejanje Perl datoteke je pomembno, da se v imenu datoteke ne nahaja presledek, pomembno pa je tudi, da ima sama datoteka končnico `.pl`.

2.4.2 Http header in Perl

Pri delu s spletnimi skriptami je pomembno, da se interpreterju Perl sporoči, da skripta deluje v spletnem brskalniku. To se stori tako, da skripta izpiše vrstico, ki se ji reče z drugim imenom HTTP header. To izgleda nekako takole:

```
#!/usr/bin/perl

print "content-type: text/html \n\n";
```

V novejših inačicah Perla je sedaj mogoče vključiti paket CGI in izpis HTTP header potem zgleda nekoliko drugače:

```
#!/usr/bin/perl

use CGI;

my $query = new CGI;
print $query->header();
```

Po uporabi HTTP headerjev je mogoče pričeti z izpisom v brskalnik, kar je potem zelo podobno kot bi delali izpis v konzolo. Program potem zgleda nekako takole:

```
#!/usr/bin/perl

use CGI;

my $query = new CGI;
print $query->header();
print "Hello, Perl!";
```

2.4.3 Izvajanje Perl skripte

Perl skripto je potrebno skopirati na spletni strežnik, če jo želimo izvajati s spletne strani. Ko je skripta prenesena na strežnik, je treba skripti spremeniti pravice z ukazom `CHMOD` in dodeliti pravico anonymous execution. V večini primerov je dovolj, da dodelimo pravico +0755.

2.4.4 Sintaksa jezika Perl

Za razliko od ostalih programskih jezikov ima Perl zelo specifično sintakso. Zelo je pomembno, da se razvije dobre sintaktične navade, da se je mogoče izogniti kasnejšemu nepotrebnemu razhroščevanju.
Imena datotek, spremenljivk in seznamov so občutljivi na velike in male črke.
Komentarji so namenjeni programerjem, da si označijo del kode. Komentar je vse, kar sledi znaku # v isti vrstici.
Za tako imenovane escape kode se v jeziku Perl uporablja poševnica nazaj "\". Te kode pišemo med dvojne narekovaje pri nizih znakov.

2.4.5 Podatkovni tipi

V jeziku Perl imamo več vrst podatkovnih tipov:

- **Skalarni tipi** so preproste spremenljivke, ki vsebujejo le eno vrednost: niz, število ali referenco. Niz lahko vsebuje katerikoli simbol, črko ali številko. Števila lahko vsebujejo eksponento, cela števila ali števila v plavajoči vejici.

```
# DEFINICIJA SPREMENLJIVK SKALARNEGA TIPA
$stevilo = 5;
$eksponenta = "2 ** 8";
$niz = "Hello, Perl!";
```

- **Seznami** so sestavljeni iz množice skalarnih tipov. Seznam lahko vsebuje neomejeno število elementov. Array deklariramo v jeziku Perl s simbolom @.

```
#DEFINICIJA NEKATERIH SEZNAMOV
@dnevi = ("Ponedeljek", "Torek", "Sreda");
@meseci = ("April", "Maj", "Junij");
```

- **Hash tabele** so kompleksni seznami sestavljeni iz parov ključ/vrednost. Spremenljivko tipa hash se definira s simbolom

```
#DEFINICIJA HASH TABEL
%kovanci = ("Quarter", 25, "Dime", 10, "Nickle", 5);
my %hash = Object();
$hash{"kljuc"} = "vrednost";
```

Nize znakov je možno definirati na več načinov. Lahko se uporabi enojne, dvojne, možno pa je uporabiti celo uporabniško določene znake.

```
$enojni = 'Ta niz ime enojne narekovaje.';
$dvojni = "Ta niz ime dvojne narekovaje.";
$uporabniskoDoloceno = q^Stresica je sedaj nas narekovaj.^;
```

Nize lahko formatiramo tudi s pomočjo tako imenovanih escape kod, ki delujejo nekako kot mini funkcije.

2.4.6 Stavki in zanke

Pogojni stavek je definiran kot:

```
if ($x == 6) {
    print "X je 6.";
}
elsif ($x == 4) {
    print "X je 4.";
}
elsif ($x == 5) {
    print "X je 5!";
}
```

For zanka se lahko uporabi na več načinov.

- Zanka, kjer ima neka spremenljivka določen razpon vrednosti

```
for($i = 1; $i < 5; $i++) {
    # natisni vsako iteracijo eno vrstico
    print "<tr><td>$i</td><td>This is row $i</td></tr>";
}
```

- Razpon vrednosti na drug način

```
for $num (1..100){
    next if $num % 2;
    print $num,"\n";
}
```

- Iteracija skozi vse elemente

```
foreach $names(@names) {
    print "<tr><td>$count</td><td>$names</td></tr>";
    $count++;
}
```

Poleg omenjenih je tu še tako imenovana while zanka.

```
# INICIALIZACIJA SPREMENLJIVKE
$iteracija = 0;
```

```
# WHILE ZANKA
while ($iteracija <= 7) {
    # IZPISI SPREMENLJIVKO IN HTML LINE BREAK
    print "$iteracija<br />";
    # POVECAJ VREDNOST SPREMENLJIVKE
    $iteracija ++;
}
```

Poleg zank obstajajo se rezervirane kontrolne besede, ki spreminjajo normalen potek zank: `next, last in redo`.

2.4.7 Preddefinirane spremenljivke

V programskem jeziku Perl obstaja nekaj preddefiniranih spremenljivk. Najpomebnejše med njimi so:

- `$$` predstavlja sistemsko številko procesa, ki jo ima instanca Perla, ki poganja trenutno skripto.
- `$0` vsebuje ime trenutno poganjanega programa
- `$a` in `$b` sta specialni spremenljivki, ki se uporabljata pri sortiranju.
- `$_` je zagotovo najbolj pomembna splošna spremenljivka, saj je uporabna na veliko načinov. Uporablja se kot privzeti vhod v funkcije in tudi kot rezultat iskanja vzorcev. Tukaj je seznam funkcij, ki imajo privzet parameter `$_`:
  `abs`, `alarm`, `chomp`, `chop`, `chr`, `chroot`, `cos`, `defined`, `eval`, `evalbytes`, `exp`, `fc`, `glob`, `grep`, `hex`, `int`, `lc`, `lcfirst`, `length`, `log`, `lstat`, `map`, `mkdir`, `oct`, `ord`, `pos`, `print`, `printf`, `quotemeta`, `readlink`, `readpipe`, `ref`, `require`, `reverse` (samo v skalarnem kontekstu), `rmdir`, `say`, `sin`, `split` (drugi argument), `sqrt`, `stat`, `study`, `uc`, `ucfirst`, `unlink`, `unpack`. Uporablja se v `foreach` funkciji, ko niso podani drugi argumenti. Pri branju iz datoteke se tudi nastavi vsebina te spremenljivke.

- `$@` je namenjena za prenosu parametrov v funkcije, kjer dostopamo do primernega argumenta na način `$_[i]`. Ta spremenljivka je privzeti niz za funkcije `push, pop, shift` in `unshift`.

Poglavje 3

Uporabljene odprtokodne rešitve

3.1 jQuery

jQuery je JavaScript knjižnica, ki omogoča spletnim razvijalcem svojim spletnim stranem dodati funkcionalnost [12]. Je odprtokodna rešitev in je dostopna brezplačno s tem, da se upošteva MIT licenca. V zadnjih letih je jQuery postala najbolj uporabljena JavaScript knjižnica pri razvoju spletnih strani.
Za uporabo jQuery je potrebno v HTML oz. na spletni strani narediti le sklic na jQuery JavaScript knižnico. Nekatere spletne strani gostujejo lokalno kopijo knjižnice, druge se sklicujejo na knjižnico, gostovane kje drugje naprimer na Google ali jQuery strežniku.
Glavni razlog za uporabo jQuery je poleg brezplačne licence združljivost z večino spletnih brskalnikov. Ker vsak spletni brskalnik prikazuje HTML, CSS in JavaScript po svoje, je spletnim razvijalcom nekoliko teže razviti spletno stran, ki bi povsod izgledala enako. Namesto, da razvijalec napiše prirejene funkcije za vsak brskalnik posebej, lahko uporabi eno samo jQuery funkcijo, ki je prenosljiva med večino brskalnikov.
Preden se uporabnik loti uporabe jQuery, je važno, da ima vsaj osnovno znanje o `HTML`, `CSS` in `JavaScript` [14].

Podrobne informacije o jQuery so dostopne na uradni spletni strani [13].

3.1.1 Kaj je jQuery?

jQuery je preprosta "piši manj, naredi več"JavaScript knjižnica. Namen jQuery je poenostavitev uporabe JavaScript na spletni strani. jQuery uporabi večino splošnih nalog, ki zahtevajo veliko vrstic JavaScript kode, in jih zapakira v eno samo metodo, ki zahteva uporabo le ene vrstice kode.
jQuery knjižnica vsebuje naslednje elemente:

- HTML/DOM manipulacija
- CSS manipulacija
- metode za HTML dogodke (events)
- efekti in animacije
- Ajax
- Pripomočki

3.1.2 Inštalacija jQuery

jQuery se lahko na več načinov uporabi na določeni spletni strani. Lahko se prenese celotno knjižnico z jQuery uradne spletne strani ali pa se uporablja jQuery, ki se nahaja na enem večjih strežnikov kot je naprimer Google.
Če se uporabnik odloči za prenos knjižnice, bo opazil, da je vse to le ena sama datoteka, na katero se potem iz HTML strani naredi le referenco.
Lokalna kopija knjižnice:

```
<head>
<script src="jquery-2.1.1.js"></script>
</head>
```

Uporaba knjižnice v spletu:
Google

```
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/2.1.1/
   jquery.min.js"></script>
</head>
```

Microsoft

```
<head>
<script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1.1.
   min.js"></script>
</head>
```

3.1.3 jQuery sintaksa

jQuery sintaksa je po meri narejena za **izbiranje** HTML elementov izvajanje **akcij** nad njimi.
Osnovna sintaksa zgleda takole: `$(selektor).akcija`

- Znak `$` za dostop do jQuery
- `(selektor)` za izbiro ali iskanje HTML elementov
- jQuery `akcija`, ki se izvede nad izbranimi elementi

Primeri:

```
$(this).hide()    - skrije trenutni element.
$("p").hide()     - skrije vse <p> elemente.
$(".test").hide() - skrije vse elemente, ki imajo class="test".
$("#test").hide() - skrije element, ki ima id="test".
```

3.1.4 Dokument ready dogodek

Dobra praksa je, če se z uporabo JavaScript in jQuery počaka, dokler se stran ne naloži do konca, saj bi v nasprotnem primeru lahko prišlo do napak, če bi naprimer hoteli skriti element, ki še ne obstaja ali pa bi hoteli dobit dimenzije slike, ki se še ni naložila.

To lahko zagotovimo tako, da prvi klic jQuery funkcij uporabimo v naslednjem bloku, ki se izvede šele, ko je stran pripravljena.

```
$(function(){

   // jQuery metode se uporabijo tukaj...

});
```

3.1.5 jQuery selektorji

jQuery selektorji so ena najpomembnejših delov jQuery knjižnice. Omogočajo izbiro in manipulacijo HTML elementov.
jQuery selektorji se uporabljajo za iskanje oz. izbiranje HTML elementov glede na njihov id, class, tip, atribut, vrednost atributa in še veliko več. Osnovani so osnovi že obstoječih CSS selektorjev, dodani pa so še nekateri drugi selektorji.
Vsi selektorji v jQuery se pričnejo s simbolom za dolar, ki mu sledita oklepaja: `$()`.
Vrste selektorjev:

- Element selektor izbere element na podlagi imena(`$("p")`)
- `#id` selektor izbere element preko njegovega id(`$("#test")`)
- .class selector izbere elemente preko class(`$(".test")`)

3.1.6 jQuery in dogodki

jQuery je napisan po meri za odziv na dogodke na HTML strani.

Kaj so dogodki?

Vse različne akcije, ki jih obiskovalec na spletni strani lahko sproži, se imenujejo dogodki.
Primeri:

- premik miške preko nekega HTML elementa
- izbira neke radio kontrole v formi
- klik na nek element

Najbolj pogosti dogodki so:

- **Miška** click, dblclick, mouseenter, mouseleave
- **Tipkovnica** keypress, keydown, keyup
- **Forma** submit, change, focus, blur
- **Dokument/okno** load, resize, scroll, unload

jQuery sintaksa za dogodke

V jQuery ima večina DOM dogodkov neko ekvivalentno jQuery metodo.
Za uporabo click dogodka na vseh odstavkih na strani uporabimo klic
`$("p").click()`
Naslednji korak je definiranje, kaj se zgodi ob določeni akciji kar storimo takole:

```
$("p").click(function(){
  // akcija pride sem
});
```

V tem koraku je določena funkcija, ki se požene ob dogodku click.

3.1.7 jQuery efekti

jQuery ima vgrajene razne efekte za delo s HTML elementi.

- **hide/show** metodi sta namenjeni skrivanju ali prikazovanju HTML elementa
- **toggle** metoda izmenično skrije oz. prikaže HTML element

- **fade** metode so namenjene, da se neki HTML element počasi prikaže ali skrije
- **slide** metode so namenjene temu, da se HTML element pripelje na svoje mesto
- **animate** metoda je namenjena, da se novi CSS stil nastavi preko animacije od prvotnega do ciljnega CSS stila

3.1.8 jQuery povratne funkcije

V jQuery lahko pride do napake, če želimo dostopati do objekta še preden se je nek efekt končal nad njim. Da se bi se izognili temu problemu, je v jQuery omogočena uporaba *povratnih* funkcij, ki se pokličejo šele po končanem efektu.
Sintaksa: `selector.hide(speed, callback);`
Primer:

```
$("button").click(function(){
  $("p").hide("slow",function(){
    alert("Odstavek je sedaj skrit.");
  });
});
```

3.1.9 jQuery veriženje ukazov

Z jQuery je možno verižno povezati več funkcij in metod. To omogoča izvajanje več jQuery metod na enem HTML elementu v enem samem stavku.
Tehnika, imenovana veriženje ukazov, omogoča izvajanje več jQuery ukazov enega za drugim na istem HTML elementu. V tem primeru ni treba brskalniku poiskati večkrat en in isti HTML element.
Veriženje se naredi tako, da se prejšnji akciji preprosto doda novo akcijo.
Primer:

```
$("#p1").css("color","red").slideUp(2000).slideDown(2000);
```

3.1.10 jQuery in Ajax

Za Ajax je v jQuery na voljo nekaj funkcij, ki omogočajo izvajanje zahtev na strežniku ne da bi bilo potrebno narediti osvežitev trenutne strani.

jQuery load() metoda

jQuery `load()` funkcija je preprosta vendar zelo močna Ajax metoda.
Load() ukaz prebere podatke s strežnika ter jih shrani v izbrani HTML element.
Sintaksa: `$(selector).load(URL,data,callback);`
`URL` je obvezni parameter in pomeni naslov strežnika od koder se prenesejo podatki. Opcijski parameter `data` predstavlja množico parov ključ/vrednost, ki se prenesejo na strežnik. Opcijski `callback` parameter predstavlja funkcijo, ki se izvede po končanem Ajax klicu.
Primer:
Datoteka `demo_test.html`

```
<h2>jQuery in Ajax je zabavno!!!</h2>
<p id="p1">To je neko besedilo v odstavku.</p>
```

Naslednji klic naloži vsebino datoteke v določen element

```
$("#div1").load("demo_test.txt");
```

Bolj kompleksen primer, kjer se uporabi povratna funkcija:

```
$("button").click(function(){
  $("#div1").load("demo_test.txt",function(responseTxt,statusTxt
      ,xhr){
    if(statusTxt=="success")
      alert("Zunanja vsebina uspesno nalozena!");
    if(statusTxt=="error")
      alert("Napaka: "+xhr.status+": "+xhr.statusText);
  });
});
```

jQuery post in get metodi

Metoda `$.get()` se uporablja za prenos podatkov s strežnika z uporabo HTTP GET zahteve.
Sintaksa: `$.get(URL,callback)`
Obvezni parameter `URL` prikazuje spletni naslov, ki se uporabi pri zahtevi podatkov.
Opcijski parameter `callback` predstavlja funkcijo, ki se izvede ob uspešno končani zahtevi.
Primer:

```
$("button").click(function(){
  $.get("demo_test.asp",function(data,status){
    alert("Data: " + data + "\nStatus: " + status);
  });
});
```

Metoda `$.post()` se uporablja za prenos podatkov s strežnika z uporabo HTTP POST zahteve.
Sintaksa: `$.get(URL,data,callback)`
Obvezni parameter `URL` prikazuje spletni naslov, ki se uporabi pri zahtevi podatkov.
Opcijski parameter `data` predstavlja podatke, ki se pošljejo poleg zahteve.
Opcijski parameter `callback` predstavlja funkcijo, ki se izvede ob uspešno končani zahtevi.
Primer:

```
$("button").click(function(){
  $.post("demo_test_post.asp",
  {
    name:"Donald Duck",
    city:"Duckburg"
  },
  function(data,status){
    alert("Data: " + data + "\nStatus: " + status);
  });
});
```

jQuery ajax metoda

Metoda `$.ajax()` se uporablja za ajax klice.

Sintaksa: `$.ajax(url:url_address, type:type, data:user_data, dataType:type_of_data, success:callback)`

Parameter `URL` prikazuje spletni naslov, ki se uporabi pri zahtevi podatkov, data predstavlja podatke za strežnik, `dataType` pa tip podatkov, ki se prenašajo.

Parameter `callback` predstavlja funkcijo, ki se izvede ob uspešno končani zahtevi.

Primer:

```
$("button").click(function(){
  $.ajax(
    url     : "demo_test.asp",
    type    : "POST",
    data    : "parameter=prvi",
    dataType: "text",

    success : function(data){
        alert("Data: " + data);
    });
});
```

3.2 jQuery Corner

jQuery corner je knjižnica, ki omogoča uporabo raznih zaobljenih robov na raznih HTML elementih [16].

Za uporabo jQuery je potrebno v HTML oz. na spletni strani narediti sklic na jQuery Corner JavaScript knižnico. Poleg tega je potrebno v SCRIPT del HTML dokumenta dodati naslednji ukaz, seveda prirejen za trenutno uporabo:

```
$('.roundedCorner').corner();
```

Več možnosti uporabe knjižnjice je predstavljeno na jQuery Corner spletni strani [17].

3.3 JqGrid

JqGrid je vtičnik za uporabe tabel in je tako del jQuery knjižnice. Je brezplačen, odprtokodni in uporablja MIT licenco.
Napisan je tako, da je prenosljiv med veliko brskalniki, ima CSS podporo ter podpira različne preobleke.
Instalacija samega JqGrida je enostavna [20]. Vse kar je potrebno, je to, da se prekopira `jquery.jqGrid.js` ter `i18n/*.*` datoteke v lokalno mapo, v izvorni datoteki pa se uporabi jqGrid in povezane datoteke:

```
<link rel="stylesheet" type="text/css" media="screen" href="css/
   ui-lightness/jquery-ui-1.10.4.custom.css" />
<link rel="stylesheet" type="text/css" media="screen" href="css/
   ui.jqgrid.css" />

<script src="js/jquery-2.1.1.js" type="text/javascript"></script
   >
<script src="js/i18n/grid.locale-en.js" type="text/javascript
   "></script>
<script src="js/jquery.jqGrid.js" type="text/javascript"></
   script>
```

Tabela podpira naslednje funkcionalnosti:

Večjezična podpora

Uporabniški vmesnik za tabele je preveden v več kot 20 jezikov.

Ostranjevanje in razvrščanje

Tabele omogočajo predstavitev podatkov na način, ko je naenkrat v tabeli prikazano le del podatkov tako imenovana stran podatkov, preko njih se je možno sprehajati preko temu namenjenim tipkam. Vse podatke je možno urediti po stolpcih, možno je tudi napisati lastno prilagojeno razvrstilno funkcijo, ki po želji razvrsti podatke.

Iskanje in filtriranje

Med vsemi podatki v tabeli je možno iskati in tudi prikazati le podatke, ki ustrezajo določenim kriterijem.

Neposredno urejanje zapisov

Podatke je možno urejati neposredno v celici, možno je tudi dinamično ustvarjanje forme za urejanje zapisa.

Podprte podstrukture

V sami tabeli je možno vgnezditi več table, podatke je možno predstaviti tudi v drevesni strukturi, s katero je bolje prikazana hierarhija podatkov.

Podpora za Ajax

Za ostranjevanje, iskanje podatkov, razvrščanje je možna uporaba Ajax metode za pridobitev podatkov s strežnika.

XML/JSON podpora

V sami tabeli so lahko podatki, katerih vrednosti se nastavijo preko XML/JSON standarda, lahko se uporabi tudi lokalni seznam podatkov.

Več podatkov o JqGrid knižnjici lahko vidimo na spletni strani [19].

3.4 CodeMirror spletni urejevalnik besedila

CodeMirror je odprtokodni urejevalnik besedila [18], ki ga lahko vključimo v spletno stran. Knjižnica ima vključeno le kodo za urejevanje besedila, brez orodnih gumbov ali kake druge funkcionalnosti uporabniškega vmesnika. Dodani so bogati API klici s pomočjo katerih je možno manjkajočo funkcionalnost dokaj enostavno dodati.

Poglavje 4

Implementacija

4.1 Namen programa

Program je namenjen urejanju JSON datotek ter odvisnosti med njimi. Potrebujemo ga, ker je veliko bolj enostavno urejati JSON datoteke preko spletnega vmesnika kot urediti vsako JSON datoteko posebej, nato pa še postaviti pravilno datotečno strukturo.

S programom lahko urejamo več ločenih JWE projektov oziroma njihove datoteke, ki so zapisane v JSON formatu. Datoteke posameznega projekta so med seboj povezane preko takoimenovanih shem, kjer so opisane lastnosti vsake JSON datoteke, med njimi je napisano tudi, kako je neka entiteta lahko del druge entitete. Vsak projekt ima svoje ime ter svojo `$root` mapo (vse datoteke enega JWE projekta so zapisane v mapi `$root` oziroma v njenih podmapah). Ob izbiri projekta se avtomatsko izbere tudi mapa, vsa imena datotek izbranega projekta so relativna gleden na `$root` (to pomeni, da, na primer, ime datoteke `PROJ-Sorting/proj/Sorting.atp` program *JSONWebEditor* avtomatsko razume kot
`$root/Projects/PROJ-Sorting/proj/Sorting.atp`.

Pot do datoteke vsebuje lahko znake {}, ki se nato nadomestijo z imenom entitete, `$i` se zamenja s podanim parametrom, ki jih poda prednik od trenutno obdelovane entitete.

Primer 1:
Imamo entiteto `Test` tipa `Algorithm`, ki je del projekta `Sorting`. V shemi entitete `Algorithm` je napisano ime datoteke:

```
"Filename" : "$1/algs/ALG-{}/{}.atal",
```

Po zamenjavi znakov program ugotovi, da gre za datoteko:

```
$root/Projects/PROJ-Sorting/algs/ALG-Test/Test.atal
```

Primer 2:
Narediti želimo nov projekt po običajni metodi. Projekti se nahajajo v mapi `/home/user/projects`. Naredili bom projekt z imenom `Test`, dodali mu bomo tri algoritme `Algoritem1, Algoritem2, Algoritem3` in dva test seta `TestSet1` in `TestSet2`.

Brez programa je prej izdelava projekta potekala takole:

1. Skrbnik naredi mapo za projekt
   `/home/user/projects/PROJ-Test/proj`
2. V prej omenjeni mapi skrbnik naredi in uredi datoteko `Test.atp`
3. Skrbnik naredi mapo za `Algoritem1`
   `/home/user/projects/PROJ-Test/algs/ALG-Algoritem1`
4. V prej omenjeni mapi skrbnik naredi in uredi datoteko `Algoritem1.atal`.
5. Skrbnik naredi mapo za `Algoritem2`
   `/home/user/projects/PROJ-Test/algs/ALG-Algoritem2`
6. V prej omenjeni mapi skrbnik naredi in uredi datoteko `Algoritem2.atal`.
7. Skrbnik naredi mapo za `Algoritem3`
   `/home/user/projects/PROJ-Test/algs/ALG-Algoritem3`
8. V prej omenjeni mapi skrbnik naredi in uredi datoteko `Algoritem3.atal`.

9. Skrbnik naredi mapo za test sete `/home/user/projects/PROJ-Test/tests`
10. V prej omenjeni mapi skrbnik naredi in uredi datoteki `TestSet1.atts` in `TestSet2.atts`.

S program *JSONWebEditor* poteka izdelava veliko hitreje:

1. Skrbnik pritisne gumb za izdelavo projekta in poda ime `Test`.
2. Odpre se stran urejanja entitete, kjer nato s pritiskom na gumbe preprosto doda algoritme `Algoritem1`, `Algoritem2`, `Algoritem3`.
3. S pritiskom na gumbe preprosto doda test seta `TestSet1`, `TestSet2`.
4. Skrbnik vpiše vsebino vseh polj za vse entitete in pritisne gumb „Shrani“.
5. Program se sprehodi po vseh poljih in preko shem poišče pravo pot datotek za posamezno entito, kreira mape in shrani vseh 6 datotek na prava mesta s pravimi vrednostmi.

Generirane datoteke so prikazane v poglavju 6.1.

Imena JWE projektov in pripadajočih map so zapisana v tekstovni konfiguracijski datoteki `config.json`, ki se ureja ročno.
Spodaj so našteti primeri JSON datotek, ki se urejajo s programom.
Konfiguracijska datoteka `config.json`

```
{
   "AT" : {
      "Projects" : [
         "Sorting",
         "Merging"
      ],
      "Id" : "AT",
      "ProjectsRoot" : "D:/Home/ProjectsRoot",
      "Name" : "WinAT"
   }
}
```

Primer projektne JSON datoteke

```
{
   "Project" : {
      "AlgorithmTPL" : "SortAbsAlgorithm",
      "TestCaseClass" : "SortTestCase",
      "Date" : "30/07/2013",
      "Algorithms" : [
         "QuickSort",
         "BubbleSort"
      ],
      "TestSetIteratorClass" : "SortTestSetIterator",
      "TestSets" : [
         "TestSet1",
         "TestSet2"
      ],
      "Description" : "Testing several sorting methods",
      "Name" : "Sorting",
      "Author" : "Tomaz"
   }
}
```

4.1.1 Opis entitete

JSON datoteke se v podanem sistemu uporabljajo za opis entitet različnih tipov, na primer, `Project`, `Algorithm`, `TestSet`, `ResultDescription`. Entitete različnih tipov imajo različne lastnosti.

JSONWebEditor zna urejati le entitete tistih tipov, za katere ima podano shemo. Sheme so zapisane v mapi `$root/Schema`. Vse datoteke v tej mapi imajo ime po tipu entitete, ki ga opisujejo, in končnico atjs (primer: `Algorithm.atjs`).

Urejanje entitete se sproži tako, da se v programu *JSONWebEditor* izbere določen projekt in nato pritisne na ikono za urejanje. Program odpre urejanje entitepe tipa `Projekt`. Pri urejanju se razčleni celotno entiteto glede na shemo in glede na soodvisnosti se na obstoječi spletni strani pojavijo v tabeli elementi sheme za projekt, zraven pa so dodani tudi elementi podshem. Na

kratko povedano, celotno urejanje vseh entitet za en projekt se ureja na eni sami spletni strani.

Primer:
Entiteta `projekt.atjs` ima lahko vključen nekaj spremenljivk tipa `string` in en tip `Entity[]`, podtip `Algorithm` po imenu Algorithms. `Algorithm` ima lahko vključene druge entitete. Program prikaže na ekran tabelo, kjer so po vrsticah razvrščene spremenljivke tipa `string`, sledi seznam entitet `Algorithms`. Za vsako entiteto lahko pritisnemo gumb urejanje in v tabeli se pojavi nova vrstica, kjer so naštete elementi entitete `Algorithm`.
Ob izbiri projekta se interno izvede klic:

```
http://localhost/jwe/jwe.pl?pId=AT&eType=Project&eName=Sorting
```

Klic začne urejanje JSON datoteke `Sorting.atp`; shema za to datoteko se nahaja v datoteki `$root/Schema/Project.atjs`. V tej atjs datoteki med drugim piše tudi:

```
"Filename" : "PROJ-{}/proj/{}.atp",
```

S tem je določeno mesto datoteke za to entiteto:

```
$root/Projects/PROJ-Sorting/proj/Sorting.atp.
```

Pri odpiranju entitete se samodejno preverijo vsi elementi pripadajoče sheme. Če naprimer entiteta `Projekt` vključuje seznam entitet tipa `Algorithm`, se vsi elementi prav tako preberejo in prikažejo na strani za urejanje. Novi elementi tipa `Entity` lahko potrebujejo dodatni parameter v obliki `$i=vrednost_i`, kjer je `i` zaporedna številka parametra. Pri branju atjs datoteke program *JSONWebEditor* avtomatsko vse pojavitve niza `$i` zamenja z `vrednost_i`.

4.1.2 Opis shem

Shema opisuje tip entitete (primer: shema `Algorithm.atjs` opisuje tip `Algorithm`). Struktura sheme je povzeta po http://json-schema.org. Primer sheme je v datoteki `Schema/Project.atjs`.

Shema je JSON datoteka, z naslednjimi lastnostmi:

Name

To je ime tipa entitete, ki ga ta shema opisuje

Filename

Opis imena datoteke, ki vsebuje entiteto tipa Name

Properties

JSON objekt, ki opisuje lastnosti pridružene entitete. Za vsako lastnost entitete obstaja v properties istoimenska lastnost. Za podrobnosti glej podpoglavje Lastnosti spodaj.

Order

Vrstni red atributov, kot so prikazani na spletni strani

Za vsako lastnost entitete obstaja v *properties* delu istoimenska lastnost. Za podrobnosti glej podpoglavje *Lastnosti objekta*.
Vsebina datoteke lahko vsebuje naslednje parametre:

- {} ... to se ob branju atjs datoteke nadomesti z imenom entitete
- `$i` ... to se ob branju atjs datoteke nadomesti z `i`-tim POST parametrom

Lastnosti objekta

Vsaka lastnosti je podana z JSON objektom. Ime lastnosti se ujema z imenom lastnosti entitete, ki jo opisuje. Vsaka lastnost objekta properties je opisana z naslednjimi lastnostmi:

- *description*... opis te lastnosti; ta opis se pojavi na html strani pred kontrolo (opis za uporabnika, da ve, kaj točno vpisuje v kontrolo)
- *type*... tip lastnosti

Poleg tega ima lahko lastnost še dodatne lastnosti, ki pa so odvisni od tipa. Nekatere med njimi so obvezne, druge opcijske (kar je prav tako razvidno iz spodnjega opisa).

Tip lastnosti

String

- **Izpis**
  TextField s preverjanjem vsebine

- **Dodatni parametri**
  pattern (optional): regularni izraz, ki opisuje dovoljene znake, ki se lahko pojavijo v tem nizu; če tega parametra ni, je vsebina lahko poljubna.

Integer

- **Izpis**
  TextField s preverjanjem vsebine

- **Dodatni parametri**
  minimum (optional; default: INT_MIN): najmanjša vrednost, ki je lahko vpisana v tem polju *maximum (optional; default: INT_MAX)*: največja vrednost, ki je lahko vpisana v tem polju

Double

- **Izpis**
  TextField s preverjanjem vsebine

- **Dodatni parametri**
  minimum (optional; default: MIN): najmanjša vrednost, ki je lahko vpisana v tem polju
  maximum (optional; default: MAX): največja vrednost, ki je lahko vpisana v tem polju

File

- **Izpis**
  TextField (readonly) + gumb Delete + gumb Upload + gumb Edit, kjer je ob praznem polju prikazan le gumb Upload, sicer sta prikazana le gumba Edit in Delete

- **Dodatni parametri**
  root (optional; default: "/"):

- **Opis**: Gumb Upload odpre OpenDialog in uporabniku omogoči izbiro ene datoteke. Po zaprtju dialoga se naloži izbrana datoteka v mapo *root*. Po nalaganju se v readonly kontrolo zapiše ime naložene datoteke. Gumb Edit odpre naloženo datoteko novi vrstici tabele in omogoči urejanje. Tam se poleg urejevalnika pojavi gumb Shrani.

Files

- **Izpis**
  TextField[] + gumb OpenFileManager

- **Dodatni parametri**
  root (optional; default: "/"):

- **Opis**
  Namen te kontrole je, da uporabniku na čimbolj eleganten način omogoči upload več datotek hkrati. Rezultat (zapis v JSON datoteki) je tabela nizov, ki opisujejo naložene datoteke. Ti nizi so lahko tudi regularni izrazi (recimo "quick*"). Z uporabo kontrole Files moramo doseči popolnoma enak rezultat kot z uporabo kontrole File [], razlika je le v načinu dela (pri velikem številu datotek bo zelo zamudno, če bo moral uporabnik vsako posebej naložiti; poleg tega bo tudi število zapisov v tabeli zelo veliko – z uporabo regularnega izraza lahko število teh zapisov precej zmanjšamo).
  Gumb OpenFileManager odpre urejevalnik datotek. Tu je možno v mapo *root* in njegove podmape naložiti eno ali več datotek. Po končanem

nalaganju je potrebno izbrati datoteke, ki se bodo uporabile v pripadajoči entiteti.
Ob kliku na OK v tem oknu se vsebinaenega vhodnega polja napolni z vrednostmi izbranih datotek. Ob kliku na Cancel pa se ne zgodi nič (datoteke se ne izberejo).

Entity

- **Izpis**
  TextField (readonly) + gumb Edit + gumb Delete
- **Dodatni parametri**
  eType(obvezen): tip entitete, ki je vezana na to kontrolo.
  parameters(opcijski): tabela parametrov, ki jih podamo ob klicu te entitete;

Parametri rešujejo morebitnih odvisnosti med entitetami. S pomočjo parametrov v podrejene entitete prenesemo nastavitve iz nadrejene entitete. Ob klicu entitete se med POST (oziroma GET) paramtere dodajo vsi parametri iz tabele parameters in sicer takole: `$1=parameter_1&...&$i=parameter_i`.

Primer uporabe

Entiteta `Algorithm` kot taka ne obstaja, saj je vedno vezana na entiteto `Project`. Pri urejanju entitete `Algorithm` je na nek način treba povedati, kateremu `Projektu` pripada (to je pomembno recimo pri razreševanju direktorijske strukture). Ob klicu entitete `Algorithm` kot prvi parameter podamo `PROJ-` in tako v `Algorithm` pripeljemo manjkajoči podatek o direktoriju. Ker v `Algorithm.atjs` piše:

```
"Filename" : "$1/algs/ALG-{}/{}.atal",
```

se datoteka, ki opisuje to entiteto, nahaja v:

```
$root/Projects/$1/algs/ALG-{}/{}.atal.
```

Ob kliku na gumb Edit se odpre vsebina entitete v novi vrstici tabele in omogoči urejanje.

Tabele[]

- **Opis**
  Če imenu tipa sledita znaka [], imamo opravka s tabelo tega tipa. Rezultat (zapis v json datoteki) je tabela (recimo [podatek1 , podatek2]). Kontrola mora omogočati brisanje in dodajanje elementov tabele. Posamezen element izpiše tako, kot se izpišejo elementi tega tipa.

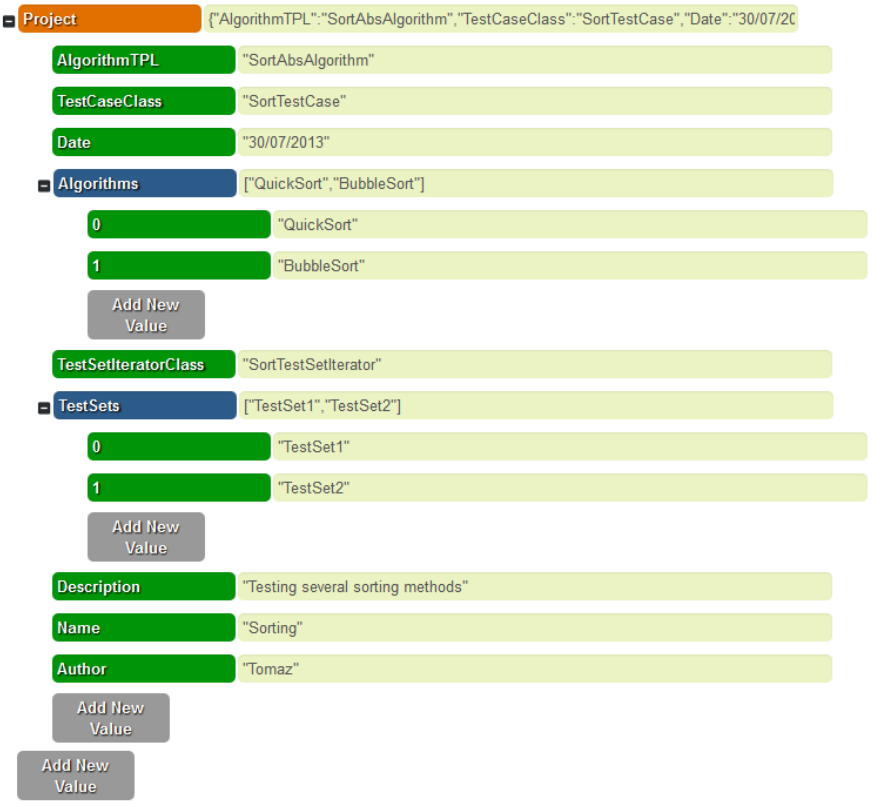
Primer
Kontrola File izpiše en TextField z gumboma Browse in Edit. Kontrola File [] prvotno izpiše le gumb Add; ob vsakem kliku na ta gumb se pojavi vrstica tipa File (TextField + gumba Browse in Edit) z dodanim gumbom Delete (ki to vrstico zbriše).

4.2 Obstoječe rešitve

Obstaja veliko obstoječih urejevalnik JSON datotek v spletnem vmesniku.

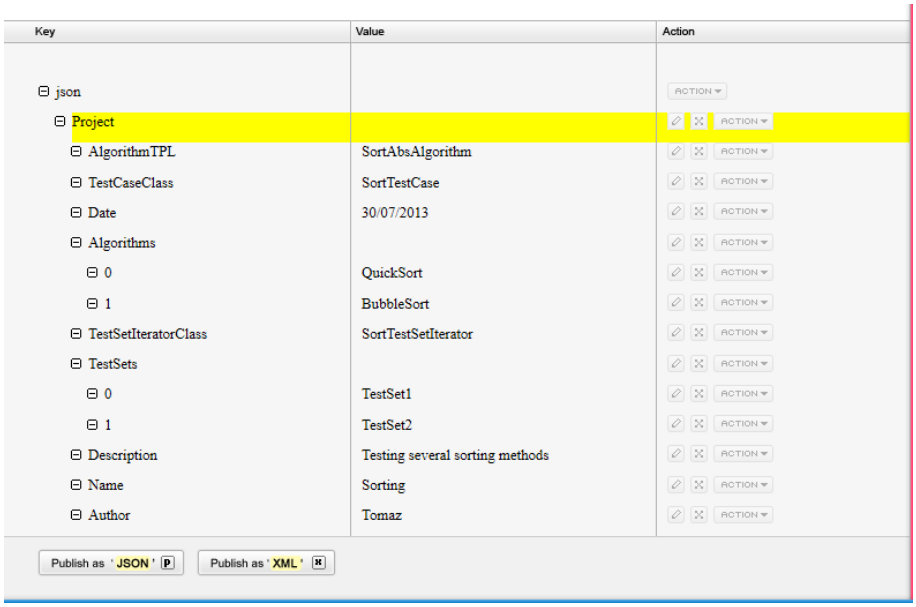
Naj jih omenimo le nekaj:

- **JSONmate** - http://jsonmate.com/
  Urejevalnik je zelo uporaben, tudi izgled je narejen v redu. Pomankljivost v primerjavi s pravkar implentiranim programom je ta, da ne omogoča urejanja povezanih datotek. Prednost je ta, da ima nekoliko lepši dizajn. Z dodano funkcionalnostjo urejanja več povezanih JSON datotek bi bil zelo uporaben editor.



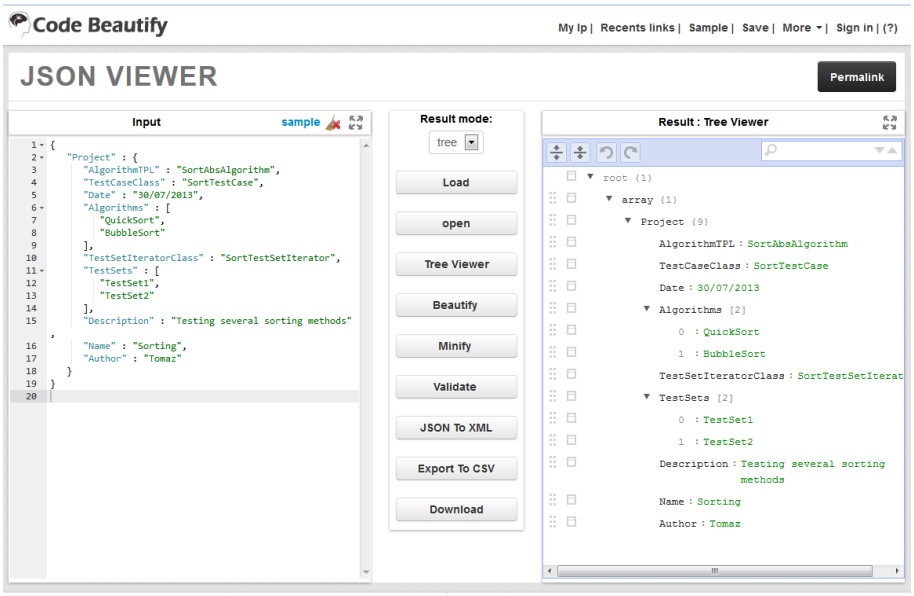
Slika 4.1: JSON mate JSON editor

- **JSON Editor** - http://jsoneditor.net/
  Editor ima sicer dokaj dobro izdelano drevesno strukturo vsebino JSON datoteke, vendar je pisava nekoliko premajhna. Ob klikanju po določenih entitetah se pojavi kričeča rumena barva. Nekoliko nerodno izbiranje dodatnih akcij. Prednost v primerjavi z JSONWebEditor je morda to, da ima dodatno funkcionalnost, slabost je malenkost slabši dizajn in nerodna izbira akcij, tudi nima možnosti urejajnja povezanih datotek.



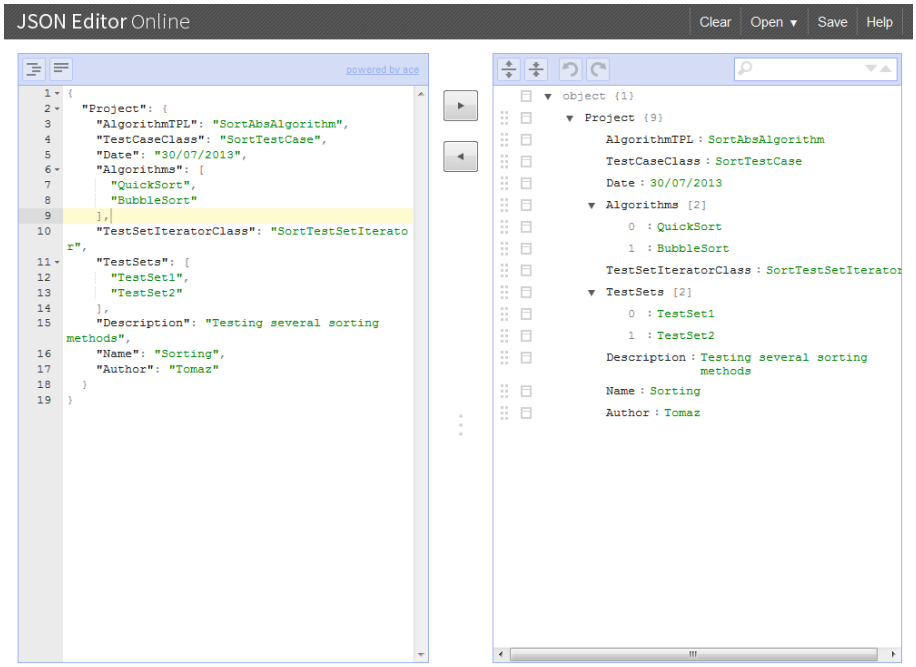
Slika 4.2: JSON editor

- **Code Beautify** - http://codebeautify.org/view/jsonviewer
  V redu JSON editor, ki ima veliko dodatnih funkcij. Urejanje JSON datotek je enostavno. Prednost pred JSONWebEditor je ta, da ima precej več funkcionalnosti, slabost je edino to, da je potrebno imeti visoko ločljivost na monitorju, da stvar v redu izgleda, ter to, da nima možnosti urejanja povezanih datotek.



Slika 4.3: Code Beautify JSON editor

- **JSON Editor online** - http://www.jsoneditoronline.org/
  Lep editor, podoben tistemu pri Code Beautify, le veliko manj ima funkcionalnosti. Prednost mogoče to, da bolj enostavno zgleda kot JSONWebEditor, slabost je ta, da se potrebuje visoka ločljivost, ter manjkajoča možnost urejanja povezanih datotek.



Slika 4.4: JSON editor online

4.3 Razvoj

4.3.1 Uporabljena orodja

Eclipse

Za razvoj je bilo uporabljeno razvojno orodje Eclipse IDE for Java Developers, verzija: Juno Service Release 2. Poleg osnovne aplikacije je dodatno se instaliran vtičnik *E.P.I.C.*, ki omogoča urejanje, razvoj in razhroščevanje Perl datotek.

Za pravilno delovanje razhroščevalnika za Perl je potrebno dodatno instalirati še en modul z ukazom:

```
cpan install PadWalker
```

Poleg omenjenega modula sta instalirana še naslednja dva modula: *Json Editor Plugin* za urejanje JSON datotek in *Eclipse Web Developer Tools*, s

katerim je precej lažje urejati spletne datoteke tipa HTML, JS, CSS,...

jQuery

Za posodabljanje vsebine spletne strani se uporablja JavaScript, nekje jQuery, ponekod pa tudi Ajax. Naprimer, ko pritisnemo gumb Briši entito, se uporabi JavaScript funkcija, ki skrije določene ikone, prikaže druge ter odstrani sedaj nepotrebne vrstice v prikazani datoteki.
jQuery se uporabi, da se pri pošiljanju podatkov preko forme na spletni strani po zaključku procesa na ekranu pojavi sporočilo, da je proces končan.
Za nalaganje datotek na strežnik, kreiranje novih datotek, izvajanje funkcij v upravljalniku datotek,... se uporablja Ajax na način kot je opisano na spletni strani [15].
Na strežniku obstaja datoteka `performAction.pl`, ki se pokliče, kadar je potrebno naložiti datoteko. Ta mora iti skozi cel seznam izbranih datotek in naložiti eno za drugo.

4.4 JqGrid in urejevalnik datotek

Za urejevalnik datotek oz. File Manager je bil uporabljen vtičnik JqGrid [19].
Pri implementaciji je bil zaznan problem, da se pri spreminjanju velikosti posameznega stolpca ne prikaže nova pozicija stolpca pravilna. Ta težava je bila odpravljena na način kot je opisano na spletni strani [21].
Dodani so bili tudi gumbi za dodatne akcije:

- Domov
- Pojdi eno mapo navzgor
- Kreiranje nove mape
- Kopiraj datoteke/mapo
- Izreži datoteke/mapo

- Prilepi datoteko
- Brisanje datoteke/mape
- Snemi datoteko
- Naloži datoteko

Za vse akcije razen kopiranje in rezanje datotek se uporablja Ajax tehnologija, ki spletnemu strežniku pošlje zahtevo za izvajanje akcij. Kopiranje in rezanje ne uporablja Ajax, ker se samo v začasni pomnilnik shrani lokacija trenutno izbrane datoteke/mape, le-to pa se uporabi pri akciji prilepi.
Za aktiviranje/deaktiviranje gumbov v orodni vrstici skrbi JavaScript.

4.5 CodeMirror spletni urejevalnik besedila

Za urejanje besedila se uporablja odprtokodni spletni urejevalnik besedila. Ta urejevalnik se uporablja pri entitetah tipa File. Privzeto urejevalnik ni prikazan za nobeno entiteto, vendar se prikaže ob pritisku na ikono, ki pomeni urejanje datoteke.

4.6 Struktura in delovanje programa

4.6.1 jwe.pl

Program najprej prebere parametre, ki jih prejme s spletne strani. Glede na parametre se potem program odloči ali bo startana ena izmed akcij kot so pregled ene sheme, pregled vseh obstoječih shem ali startanje upravljalnika datotek (FileManagerja), prikaz vseh glavnih projektov, prikaz projektov ali urejanje posameznega projekta.
S pomočjo pomožnega modula se nato prebere pot, kjer se nahaja HTML predloga, ki bo uporabljena. Nato se s podane poti prebere HTML predloga, potrebni parametri za HTML predlogo se preberejo s pomočjo neke funkcije

v pomožnem modulu. Generirana stran se nato prikaže na uporabnikovem računalniku.

4.6.2 performAction.pl

Datoteka se uporablja pri vseh akcijah, ki se zgodijo na spletni stran. Program preveri vhodne parametre in se glede na njihovo vsebino odloči za primerno akcijo.
Trenutno podprte akcije so naštete spodaj.

Branje datoteke

Ta akcija prebere vsebino datoteke na spletni strani in jo vrne spletni strani, ki jo je zahtevala.

Pisanje datoteke

Ta akcija prebere vsebino urejene datoteke na spletni strani in jo shrani.

Izpis seznama datotek v neki mapi

Pri uporabi urejevalnika datotek je potrebno prebrati vsebino neke mape, da se lahko prikaže v tabeli. To je narejeno preko trenutno omenjene akcije.

Kreiranje nove mape

Urejevalnik datotek omogoča ustvaranje nove mape, ki to naredi preko akcije.

Brisanje datoteke/mape

Znotraj urejevalnika datotek je poleg vseh akcij možno tudi brisati datoteke in mape.

Prilepi datoteko

Po tem, ko je neka datoteka izbrana za kopiranje oz. rezanje, je seveda pomemben del to, da jo je potem možno prilepiti v drugo mapo. Če je bila datoteka izbrana za rezanje, se datoteka odstrani z izvorne mape.

Naloži datoteko v upravljalniku datotek

V upravljalniku datotek je možno tudi nalagati datoteke v trenutno izbrano mapo. Če datoteka že obstaja, se povozi z novo datoteko.
Primer implementacije nalaganja datotek.

```
my @fileHandleList = $query->upload('files');
my $uploadFolder   = $query->param('uploadFolder');

foreach my $fh (@fileHandleList)
{
    # undef may be returned if it's not a valid file handle
    if (defined $fh )
    {
        # Upgrade the handle to one compatible with IO::Handle:
        my $io_handle = $fh->handle;

        open( OUTFILE, '>', $uploadFolder . $fh );

        while ( my $bytesread = $io_handle->read( my $buffer,
            1024 ) )
        {
            print OUTFILE $buffer;
        }
    }
}
```

Vsebina HTML datoteke, ki sproži nalaganje datotek na strežnik.

```
<form id="uploadFilesForm" action="performAction.pl" method='
   post' enctype="multipart/form-data">
    <button id="uploadFileButton" type="submit">Upload</button>
    <input type="file" id="files" name="files" multiple />
```

```
    <input type="hidden" id="uploadFolder" name="uploadFolder"
       value="C:\temp" />
</form>
```

Naloži datoteko na strani za urejanje projekta

Urejevalnik entitet omogoča nalaganje datotek za polja tipa `File`. Program naloži izbrano datoteko v mapo, ki je določena preko parametra `root`, za tip *Files*.

Validacija teksta v vnosnem polju

Program omogoča validacijo vhodnih polj pri urejanju entitet. Veljavni vzorec je shranjen v parametru `pattern` za atribute tipa `String`. V primeru neuspešne validacije se neveljavno polje obarva rdeče. Validacija poteka preko Ajax protokola.

Shrani urejeno entiteto

Po končanem urejanju entitete je seveda zelo pomembno to, da se nova vsebina shrani. To uporabnik naredi s pritiskom na gumb Shrani. Akcija nato vse parametre (vsebina celotne tabele urejevalnika entitete), ki so se prenesli preko obrazca, razvrsti po vseh Entitetah in podentitetah ter shrani vse vključene datoteke.

Dodaj entiteto

Akcija kreira novo entiteto, hkrati pa generira novo HTML kodo, ki se vključi v obstoječo stran.

Briši projekt

Akcija briše projekt s seznama projektov.

4.6.3 Datoteka modules/EntityTools.pm

Datoteka vsebuje vse pomožne funkcije za dostop do konfiguracijskih datotek in do vsebine JSON datotek.

4.6.4 Datoteka modules/JsonHelper.pm

Datoteka vsebuje splošne funkcije za delo z JSON datotekami kot so branje, pisanje JSON datoteke ter branje posamezne vrednost iz JSON podatkovne strukture.

4.6.5 Datoteka modules/Tools.pm

Datoteka vsebuje splošno uporabne funkcije kot so funkcije za branje datoteke, branje vsebine datoteke v seznam nizov, preverjanje poverilnic, preverjanje id seje, ...

4.6.6 Opis delovanja

Sama implementacija oziroma sestava strani je sestavljena iz več delov. Del kode je napisano v programskem jeziku Perl, del spletne strani predstavljajo HTML predloge, del je napisan v programskem jeziku JavaScript, del kode je implementiran v jQuery.

4.6.7 Težave pri pisanju kode

Prvotna različica programa je omogočala urejanje vsake entitete posebej.
Po odločitvi, da se vse prestavi na eno stran, je bilo potrebno poskrbeti, da je vsaka kontrola na strani dobila nek unikaten id.
Vsak vhodni element v tabeli, ki je hkrati tudi obrazec(*form*), mora imeti svoj unikaten id, ki je sestavljen iz več delov, kjer prvi del številk predstavlja starše, zadnja številka predstavlja sam id trenutne kontrole. Na podlagi prednikov se vhodni podatek dodeli pravi entiteti, ki vsebino temu primerno shrani. Za dodajanje nove entiteto na obstoječi strani je potrebno vedeti

katerega tipa je entiteta in tudi kateri lastnosti pri staršu pripada. Ti dve informaciji sta shranjeni kot del id vhodnega elementa.

Za posodobljanje spletne strani, ne da bi naredili osvežitev strani, je potrebno poskrbeti, da ima nova vrstica nastavljen id 999 in nato se preko njegovega zadnje uporabljenega brata(sibling) ugotovi id in poveča za 1. Novi id nato Perl skripta uporabi za generiranje HTML kode, ki jo JavaScript po končanem Ajax klicu vključi na primerno mesto v tabeli. Po vključitvi nove kode v tabelo se ne izvedejo koraki, napisani v JavaScript delu HTML kode, ki so potrebni, da spletni urejevalnik deluje. Za to je potrebno poskrbeti posebej. Program pregleda vse kontrole, če id kontrole za spletni urejevalnik besedil obstaja, nakar se požene določen del kode.

Implementacija vgnezdenih entitet (entiteta, ki vsebuje `entity[]`, kjer vsak lahko vsebuje zopet `entity[]`, ter tako dalje...) je kar zahtevna. Najprej je potrebno HTML predlogo razbiti na več delov, kjer je vsak del namenjen urejanju enega tipa lastnosti. Za vsako podprto globino je potrebno narediti novi dve HTML datoteki, ki omogočata prikaz `Entity[]` objekta. `HTML::Template` namreč ne omogoča rakurzivnih funkcij, zato je bilo potrebno najti nov način.

4.7 Končni produkt

Po nekaj mesecih implementacije je produkt končno pripravljen za uporabo. Vse morebitne težave se bodo reševale sproti, ko bo produkt v uporabi.

Program je namenjen pregledu shem entitet. S programom je možno startati spletni urejevalnik datotek, kjer se lahko datoteke naloži, briše, kreira mape. Z njim je možno narediti nov projekt, urediti obstoječi projekt, kjer lahko vse zahtevane datoteke uporabnik naloži med urejevanjem projekta ne da bi zapustil stran. Med urejanjem entitete je prav tako možno s spletnim urejevalnikom besedila popraviti vsebino naložene datoteke.

4.7.1 Vsebina produkta

Produkt je sestavljen iz več delov, ki skupaj sestavljajo celoto in so neločljivo povezani en z drugim.

Perl datoteke

Projekt vsebuje dve gavni in tri pomožne Perl datoteke, ki omogočajo delovanje spletne strani. Datoteka `jwe.pl` skrbi za delovanje strani, datoteka `performAction.pl` opravlja vse Ajax in ostale akcije, ki jih spletna stran zahteva. Datoteka `modules/JsonHelper.pm` vsebuje funkcije za delo z Json datotekami, `modules/EntityTools.pm` ima funkcije za delo z entitetami, ki se uporabljajo v obeh glavnih Perl datotekah, `modules/tools.pm` ima samo splošne funkcije za delo z datotekami ter uporabne funkcije.

Konfiguracijska datoteka

Datoteka `Configuration/config.json` vsebuje informacije o obstoječih projektih. Tu so našteti vključeni projekti, id in ime skupine projektov ter glavna pot, kjer se projekti nahajajo.

```
{
   "AT" : {
      "Projects" : [
         "Sorting",
         "Merging",
         "TestJWE"
      ],
      "Id" : "AT",
      "ProjectsRoot" : "D:/Home/ProjectsRoot",
      "Name" : "WinAT"
   }
}
```

JavaScript

Za delovanje spletne strani je odgovornih med drugim tudi nekaj JavaScript datotek, ki se uporabljajo za dodajanje entitet, brisanje entitet, odpiranje strani. Za odpiranje strani se uporablja pošiljanje podatkov s "post"metodo, ki je implementirana kot je napisano v [11].
Funkcija odpre spletno stran tako, da začasno doda na trenutno stran nek obrazec, za pošiljanje parametrov se uporabijo začasno kreirana vhodna polja, ki jih uporabi obrazec. Funkcija nato simulira pritisk gumba *Submit*, ki nato vse podatke pošlje na strežnik, ki nato generira novo stran.

```
function postwith (to,p) {
  var myForm = document.createElement("form");
  myForm.method="post" ;
  myForm.action = to ;
  for (var k in p) {
    var myInput = document.createElement("input") ;
    myInput.setAttribute("name", k) ;
    myInput.setAttribute("value", p[k]);
    myForm.appendChild(myInput) ;
  }
  document.body.appendChild(myForm) ;
  myForm.submit() ;
  document.body.removeChild(myForm) ;
}
```

Zelo pomembna datoteka je `jquery-2.1.1.js`, ki omogoča jQuery. Datoteka `jquery.corner.js` omogoča uporabo zaobljenih robov. S pomočjo `jquery.jqGrid.js` in `i18n/*.js` je implementiran spletni urejevalnik datotek. Datoteka `jwe.js` ima vključene vse JavaScript in jQuery metode, ki se uporabljajo na spletnih straneh.

CSS datoteke

Za izgled strani skrbijo CSS datoteke.
Datoteke `ui-lightness/*.*` in `ui.jqgrid.css` sta namenjeni izgledu urejevalnika datotek. Datoteka `jwe.css` skrbi za celoten izgled spletne

strani.

Images

V tej mapi so shranjene vse slike, ki se uporabljajo na spletni strani.

codemirror

Mapa vsebuje celoten paket urejevalnika besedil CodeMirror, ki se uporablja v urejevalniku entitet.

Sheme

Sheme se nahajajo v paketu `ProjectsRoot/Schema`. To so sheme, ki opisujejo lastnosti entitet znotraj nekega projekta, zato morajo biti vedno shranjene v mapi `root/Schema`.

HTML predloge

HTML predloge se nahajajo v paketu `ProjectsRoot/Conf`. To so predloge, ki se uporabljajo za urejanje entitet znotraj nekega projekta, zato morajo biti vedno shranjene v mapi `root/Conf`.

4.7.2 Namestitev in konfiguracija produkta

Za delovanje strani je potrebno urediti več stvari. Postavljen mora biti *Apache* spletni strežnik, na sistemu mora biti instaliran *Perl*, ki mora imeti instalirane dodatne module, sicer program ne more delovati, deli programa moraji biti shranjeni na pravo pot, pot mora biti pravilno nastavljena.

Apache

Da bi pričela spletna aplikacija delovati, je potrebno prekopirati vse datoteke v mapo za spletni strežnik v podmapo z imenom `jwe`, kar na unix sistemih pomeni:

```
/var/www/jwe/
```

Za Windows okolje lahko uporabimo naslednjo pot:

```
c:\Program Files (x86)\Apache Group\Apache2\htdocs\jwe\
```

Datoteke za delovanje spletne strani seveda lahko shranimo kamorkoli, vendar moramo v tem primeru primerno spremeniti nastavitve spletnega strežnika. Projektne datoteke lahko shranimo kamorkoli, saj potem to pot napišemo v konfiguracijski datoteki.
V konfiguraciji spletnega strežnika Apache2 [3] je potrebno dodati naslednje vrstice kode:

```
ScriptAlias /jwe/ /var/www/jwe/
<Directory "/var/www/jwe">
    AllowOverride None
    Options +ExecCGI +MultiViews +SymLinksIfOwnerMatch
    Order allow,deny
    Allow from all
    AddHandler cgi-script .cgi .pl
    AddHandler default-handler .html .jpg .gif .js .txt .bat .
       css .png
</Directory>
```

Če stran še vedno ne deluje v Unix okolju, je potrebno spremeniti atribute izvajanih datotek z ukazom:

```
chmod +705 *.pl
```

Nato je treba spletni strežnik skonfigurirati tako, da bo uporabnik imel dostop do mape, kjer se *JSONWebEditor* nahaja. Podatki s konfiguracijskimi datotekami za projekte morajo tudi biti skopirani na pot, do koder ima spletni strežnik pravico brati in pisati.
Datoteki `jwe.pl` in `performAction.pl` morata imeti pravico za izvajanje.

Perl in HTML Template

V programskem jeziku Perl je napisan glavni program `jwe.pl`, ki skrbi za generiranje spletne strani, `performAction.pl`, ki se uporablja za vse akcije,

ki se dogajajo na spletni strani, ter trije pomožni moduli, ki se uporabljajo za uporabo JSON datotek, analizo JSON datotek ter uporabo nekaj preprostih funkcij.
Prepričati se moramo, da imamo za Perl instalirana modula `HTML::Template`, ki omogoča delovanje predlog, ki so uporabljene v diplomski nalogi ter modul `URI::Escape`, ki skrbi za dekodiranje nizov, poslanih preko spleta.
Modula instaliramo z ukazoma v komandni vrstici:

```
cpan install HTML::Template
cpan install URI::Escape
```

To je enako tako za Windows kot tudi za Unix okolje.

Nastavitve v samem produktu

V datotekah `jwe.pl` in `performAction.pl` je potrebno prevereti, da imata v prvi vrstici napisano pravilno pot do Perl interpreterja.
Za Windows sisteme je ta pot običajno napisana kot:

```
#!C:\Perl\Bin\Perl.exe -w
```

Za Unix sisteme je pravilna pot:

```
#!/usr/bin/perl -w
```

V datoteki `config.json`, ki se nahaja v mapi `jwe/Configuration/config.json` mora biti tudi nastavljena pravilna pot do osnovne mape, kjer se nahaja glavni projekt.
Ne eno lokacijo je potrebno skopirati še sheme in HTML predloge. To je potrebno dati v neko mapo, ki bo potem `root` za vse projekte.
To `root` mapo je nato potrebno napisati v konfiguracijski datoteki `config.json`.

Primer:
Vse datoteke smo shranili v `/var/www/jwe`.
Naredimo pot `/home/user/projectsRoot`.
Sheme shranimo v `/home/user/projectsRoot/Schema`, HTML predloge v `/home/user/projectsRoot/Conf`.

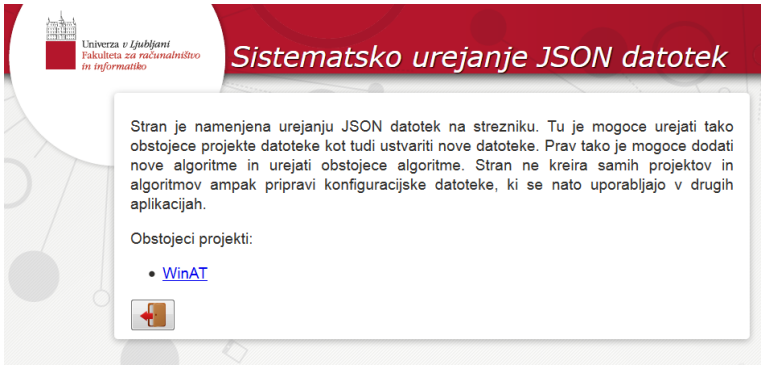
Sami projekti in ostale datoteke se bodo potem shranjevale v `/home/user/projectsRoot/Projects`.
V datoteki `/var/www/jwe/Configuration/config.json` je potrebno spremeniti eno vrstico v `"ProjectsRoot":"/home/user/projectsRoot",` .

4.8 Prikaz uporabe

Ko se program požene, se pojavi prijavno okno, kamor vnesemo prave poverilnice.

Slika 4.5: Prijavno okno

Po vnosu uporbniškega imena in gesla se pojavi okno za izbiro aktivne skupine projektov.



Slika 4.6: Seznam skupin projektov

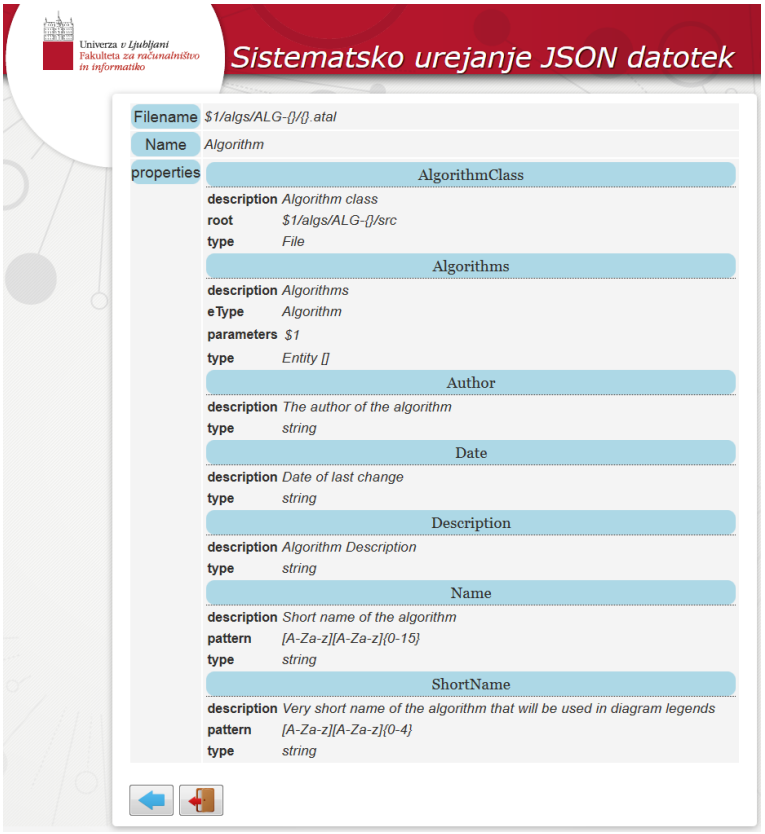
Po izbiri skupine projektov se prikaže stran s seznamom aktivnih projektov.

Slika 4.7: Seznam projektov

Sedaj se lahko odločimo za vpogledom na sheme. Najprej pritisnemo na gumb za ogled shem in pojavi se seznam obstojecih shem na sistemu.

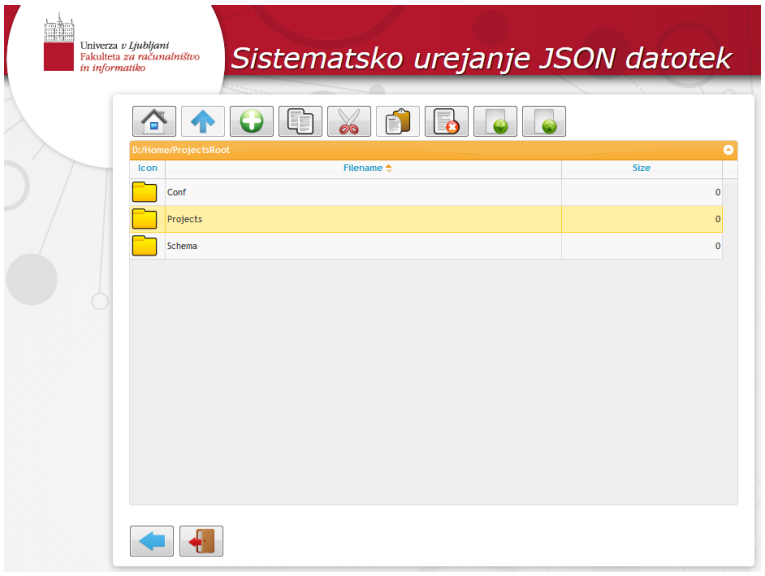
Slika 4.8: Seznam obstoječih shem

Po izbiri sheme dobimo grafični prikaz izbrane sheme za neko entiteto.



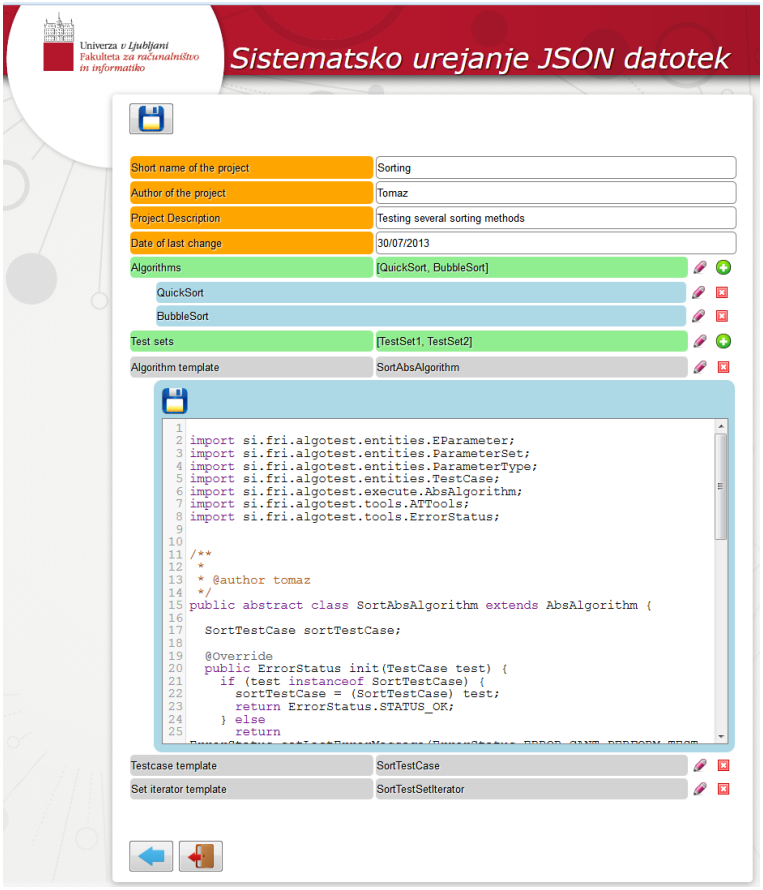
Slika 4.9: Ogled sheme

Na strani s prikazanimi projekti lahko startamo urejevalnik datotek.



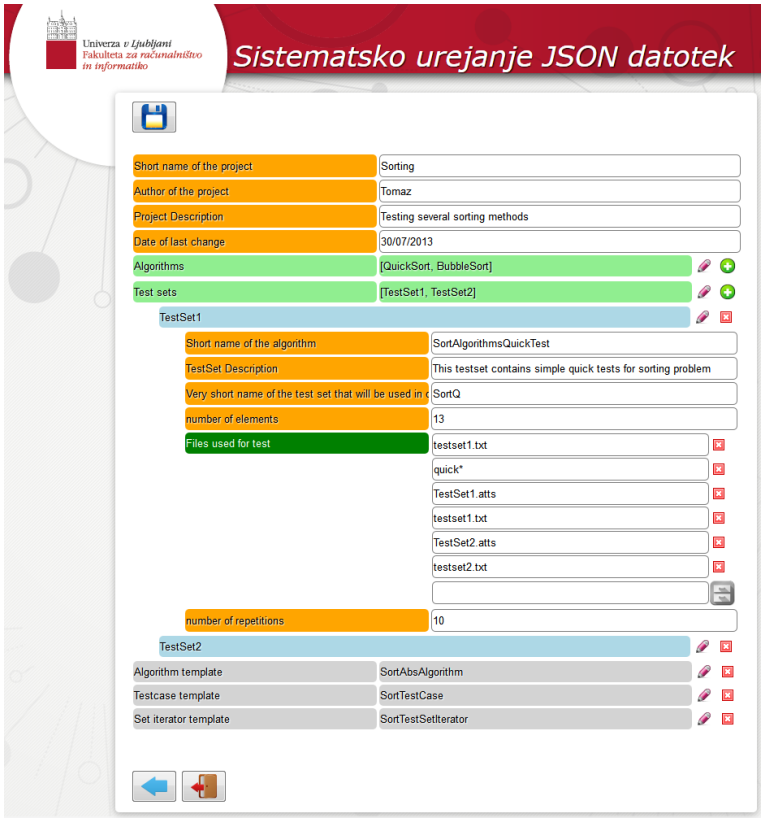
Slika 4.10: Urejevalnik datotek

Imamo možnost urejanja/brisanjem/dodajanja projekta. Po izbiri gumba za urejanje nekega projekta se odpre stran za urejanje projektne JSON datoteke.



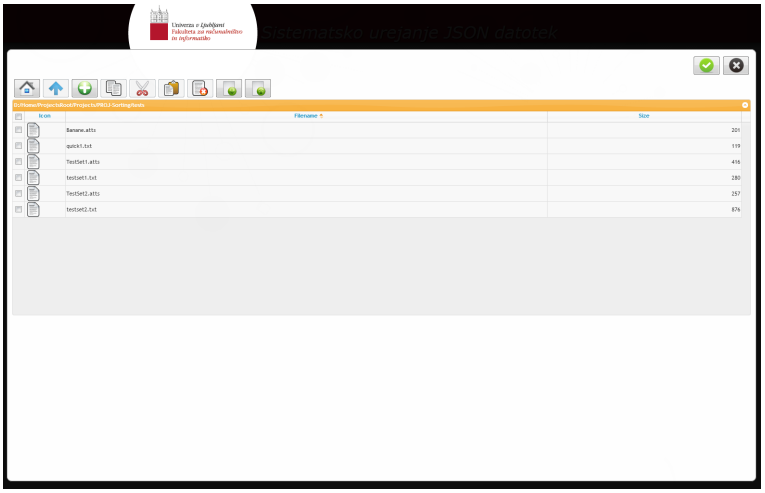
Slika 4.11: Urejanje entitete

Če ima shema tako nastavljeno, se znotraj entitete Projekt lahko ureja druga entiteta TestSet.



Slika 4.12: Urejanje entitete TestSet

Entiteta Files ima možnost startanja urejevalnika datotek, kjer se izbere datoteke.



Slika 4.13: Urejevalnik datotek pri urejanju entitete

Poglavje 5

Zaključek

Izpolnjene so bile glavne zahteve diplomskega dela. Urejati je mogoče JSON datoteke, povezane v celoto, na eni sami spletni strani. Možno je dodati in brisati entitete brez zapuščanja strani. Dodan je urejevalnik datotek, kjer je možno izvajati razne manipulacije nad datotekami in mapami kot so kreiranje mape, kopiranje, brisanje datotek, ...

V primerjavi z drugimi urejevalniki JSON datotek je implementirana rešitev boljša, ker omogoča urejanje več datotek hkrati, ki so med seboj povezane s t.i. shemami. Sheme opisuje lastnosti vseh polj JSON datotek ter tako omogočajo tudi sprotno validacijo vhodnih polj.

Namesto JSON standarda bi lahko uporabili standard YAML [7], ki je podoben JSON, le namesto oklepajev uporablja predvsem zamike in ločila. Za HTML predloge bi lahko uporabili tudi kak drugi obstoječi sistem, možno bi bilo pa tudi razviti svoj lastni sistem, ki bi bil nekoliko lažji za uporabo.

Celotno spletno aplikacijo bi bilo možno izdelati v .NET tehnologiji npr. v programskem jeziku C#. Razvoj in razroščevanje aplikacije bi bilo nekoliko lažje, a bi stran lahko gostovala le na Windows strežniku.

Pomembna izboljšava bi bila spletna avtentikacija uporabnikov. Neavtorizirani uporabniki bi npr. lahko le pregledovali JSON datoteke, ne bi jih pa mogli urejati. Kreator entitete in člani posebne skupine bi lahko entitete urejevali brez težav.

Poglavje 6

Dodatek

6.1 Uporabljene datoteke iz primera

6.1.1 Shema za projekt

Iz same sheme se da razbrati, da bo nov projekt imel pot: `PROJ-{}/proj/{}.atp`. Ker smo dali ime projekta `Test`, se bo uporabila pot: `PROJ-Test/proj/Test.atp`. Za parametre pri dodajanju algoritmov in test setov se v našem primeru uporabi `PROJ-{}`, kar *JSONWebEditor* pretvori v `PROJ-Test`.

```
{
  "Name"     : "Project",
  "Filename" : "PROJ-{}/proj/{}.atp",
  "Order"    : ["Name", "Author", "Description", "Date", "
     Algorithms", "TestSets", "AlgorithmTPL", "TestCaseClass", "
     TestSetIteratorClass"],

  "properties":   {
      "Name" : {
            "description": "Short name of the project",
            "type": "string",
            "pattern": "[A-Za-z][A-Za-z]{0-15}"
      },
      "Author" :  {
            "description": "Author of the project",
```

```
            "type": "string"
        },
        "Description" :  {
            "description": "Project Description",
            "type": "string"
        },
        "Date" : {
            "description": "Date of last change",
            "type": "string"
        },
        "Algorithms" : {
            "description" : "Algorithms",
            "type" : "Entity []",
            "eType": "Algorithm",
            "parameters" : ["PROJ-{}"]
        },
        "TestSets" : {
            "description":  "Test sets",
            "type":  "Entity []",
            "eType": "TestSet",
            "parameters" : ["PROJ-{}"]
        },
        "AlgorithmTPL" : {
            "description": "Algorithm template",
            "type": "File",
            "root": "PROJ-{}/proj/src"
        },
        "TestCaseClass" : {
            "description": "Testcase template",
            "type": "File",
            "root": "PROJ-{}/proj/src"
        },
        "TestSetIteratorClass" : {
            "description": "Set iterator template",
            "type": "File",
            "root": "PROJ-{}/proj/src"
        }
    }
```

```
}
```

6.1.2 Shema za algoritem

Iz same sheme se da razbrati, da bo nov algoritem imel pot: `$1/algs/ALG-{}/{}.atal`. Ker smo dali za parameter pri dodajanju algoritmov od projekta dobili `PROJ-Test`, za ime algoritma pa izbrali `Algoritem1`, `Algoritem2` in `Algoritem3`, se pot algoritmov pretvori v:
`PROJ-Test/algs/ALG-Algoritem1/Algoritem1.atal`
`PROJ-Test/algs/ALG-Algoritem2/Algoritem2.atal`
`PROJ-Test/algs/ALG-Algoritem3/Algoritem3.atal`

```
{
  "Name"     : "Algorithm",
  "Filename" : "$1/algs/ALG-{}/{}.atal",
  "Order"    : ["Name", "Description", "Date", "ShortName", "
     Author", "AlgorithmClass", "Algorithms"],

  "properties":   {
      "Algorithms" : {
            "description" : "Algorithms",
            "type" : "Entity []",
            "eType": "Algorithm",
            "parameters" : ["$1"]
      },
      "Name" : {
            "description": "Short name of the algorithm",
            "type": "string",
            "pattern": "[A-Za-z][A-Za-z]{0-15}"
      },
      "Description" :  {
            "description": "Algorithm Description",
            "type": "string"
      },
      "Date" : {
```

```
            "description": "Date of last change",
            "type": "string"
      },
      "ShortName" : {
            "description": "Very short name of the algorithm
               that will be used in diagram legends",
            "type": "string",
            "pattern": "[A-Za-z][A-Za-z]{0-4}"
      },
      "Author" : {
            "description": "The author of the algorithm",
            "type": "string"
      },

      "AlgorithmClass" : {
            "description": "Algorithm class",
            "type": "File",
            "root": "$1/algs/ALG-{}/src"
      }
  }
}
```

6.1.3 Shema za test set

Iz same sheme se da razbrati, da bo nov test set imel pot: `$1/tests/{}.atts`. Ker smo dali za parameter pri dodajanju algoritmov od projekta dobili `PROJ-Test`, za ime test setov pa izbrali `TestSet1` in `TestSet2`, se pot test setov pretvori v:
`PROJ-Test/tests/TestSet1.atts`
`PROJ-Test/tests/TestSet2.atts`

```
{
  "Name"     : "TestSet",
  "Filename" : "$1/tests/{}.atts",
  "Order"    : ["Name", "Desc", "ShortName", "N", "TestSetFiles
     ", "TestRepeat"],
```

```
  "properties":   {
      "Name" : {
            "description": "Short name of the algorithm",
            "type": "string",
            "pattern": "[A-Za-z][A-Za-z]{0-15}"
      },
      "Desc" :  {
            "description": "TestSet Description",
            "type": "string"
      },
      "ShortName" : {
            "description": "Very short name of the test set that
                will be used in diagram legends",
            "type": "string",
            "pattern": "[A-Za-z][A-Za-z]{0-4}"
      },
      "N" : {
            "description": "number of elements",
            "type": "string"
      },
      "TestSetFiles" : {
            "description": "Files used for test",
            "type": "Files",
            "root": "$1/tests"
      },
      "TestRepeat" : {
            "description": "number of repetitions",
            "type": "string"
      }
  }
}
```

6.1.4 Projekt

Iz primera je razvidno, da projekt vključuje algoritme `Algoritem1`, `Algoritem2` in `Algoritem3`. Poleg tega vsebuje tudi test seta `TestSet1` in `TestSet2`.

Pot: `PROJ-Test/proj/Test.atp`

```
{
   "Project" : {
      "AlgorithmTPL" : "SortAbsAlgorithm",
      "TestCaseClass" : "SortTestCase",
      "Date" : "7. junij 2014",
      "Algorithms" : [
         "Algoritem1",
         "Algoritem2",
         "Algoritem3"
      ],
      "TestSetIteratorClass" : "SortTestSetIterator",
      "TestSets" : [
         "TestSet1",
         "TestSet2"
      ],
      "Description" : "Opis",
      "Name" : "Ime",
      "Author" : "Avtor"
   }
}
```

6.1.5 Algoritem1

Pot: `PROJ-Test/algs/ALG-Algoritem1/Algoritem1.atal`

```
{
   "Algorithm" : {
      "ShortName" : "Al1",
      "AlgorithmClass" : "BubblesortSortAlgorithm",
      "Name" : "Algo1",
      "Description" : "Algoritem1",
      "Date" : "7. junij 2014",
      "Author" : "Marko",
      "Algorithms" : []
   }
}
```

6.1.6 Algoritem2

Pot: `PROJ-Test/algs/ALG-Algoritem2/Algoritem2.atal`

```
{
   "Algorithm" : {
      "ShortName" : "Al2",
      "AlgorithmClass" : "QuicksortSortAlgorithm",
      "Name" : "Algo2",
      "Description" : "Algoritem2",
      "Date" : "7. junij 2014",
      "Author" : "Marko",
      "Algorithms" : []
   }
}
```

6.1.7 Algoritem3

Pot: `PROJ-Test/algs/ALG-Algoritem3/Algoritem3.atal`

```
{
   "Algorithm" : {
      "ShortName" : "Al3",
      "AlgorithmClass" : "BubblesortSortAlgorithm",
      "Name" : "Algo3",
      "Description" : "Algoritem3",
      "Date" : "7. junij 2014",
      "Author" : "Marko",
      "Algorithms" : []
   }
}
```

6.1.8 TestSet1

Pot: `PROJ-Test/tests/TestSet1.atts`

```
{
   "TestSet" : {
      "TestRepeat" : "10",
```

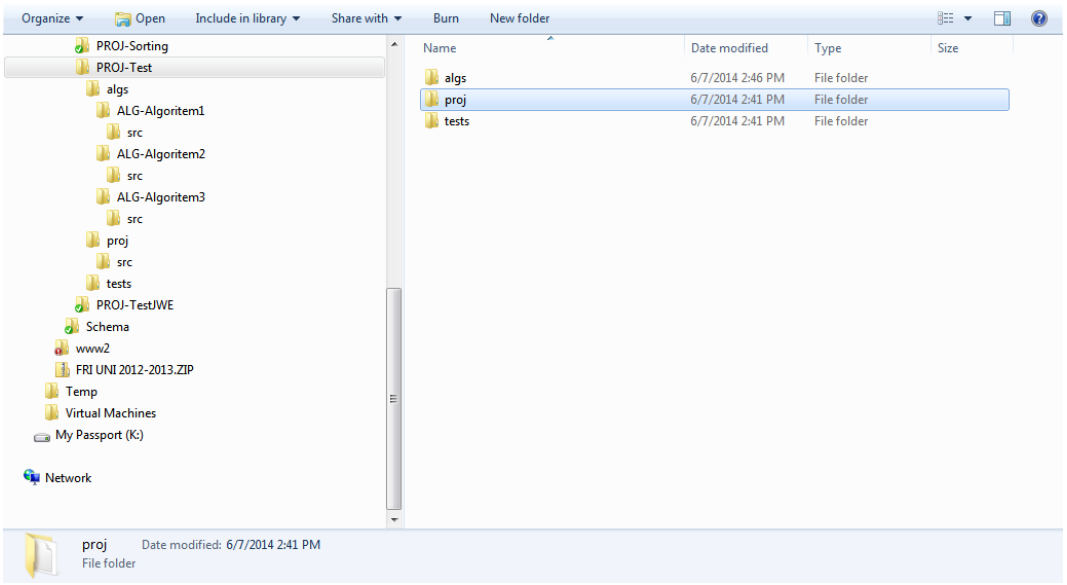
```
      "ShortName" : "TS1",
      "N" : "10",
      "TestSetFiles" : [
         "test.txt"
      ],
      "Desc" : "TestSet1",
      "Name" : "TestSet1"
   }
}
```

6.1.9 TestSet2

Pot: `PROJ-Test/tests/TestSet2.atts`

```
{
   "TestSet" : {
      "TestRepeat" : "11",
      "ShortName" : "TS2",
      "N" : "11",
      "TestSetFiles" : [
         "test2.txt"
      ],
      "Desc" : "TestSet2",
      "Name" : "TestSet2"
   }
}
```

6.1.10 Nastala datotečna struktura



Slika 6.1: Nastala datotečna struktura po kreiranju projekta

Literatura

[1] E. Robson, E. Freeman. *Head First HTML and CSS, 2nd (second) Edition*. O'Reilly Media, 2012.

[2] D. Flanagan. *JavaScript The Definitive Guide*. O'Reilly , 1998.

[3] B. Laury, P. Laury. *Apache The Definitive Guide*. O'Reilly & Associates, Inc., 1997.

[4] L. Wall, T. Christiansen & J. Orwant. *Programming Perl*. O'Reilly , 2000.

[5] (2014) Web Development Site. Dostopno na: http://www.w3schools.com/

[6] The official JSON Web Site. Dostopno na: http://www.json.org/

[7] The official YAML Web Site. Dostopno na: http://www.yaml.org/

[8] (2014) AJAX Tutorial. Dostopno na: http://www.tutorialspoint.com/ajax/index.htm

[9] (2014) What is JavaScript? Dostopno na: http://what-is-what.com/what_is/javascript.html

[10] (2014) JavaScript tutorial. Dostopno na: http://www.w3schools.com/js/

[11] (2008) Using JavaScript to POST data between pages. Dostopno na: http://mentaljetsam.wordpress.com/2008/06/02/using-javascript-to-post-data-between-pages/

[12] (2014) jQuery definition. Dostopno na: http://www.techterms.com/definition/jquery

[13] (2014) jQuery knjižnica. Dostopno na: http://code.jquery.com/

[14] (2014) jQuery tutorial. Dostopno na: http://www.w3schools.com/jQuery/default.asp

[15] (2014) AJAX-Like File Uploads with jQuery. Dostopno na: http://www.peachpit.com/articles/article.aspx?p=1766159

[16] (2014) My Favorite Rounded Corner Solutions. Dostopno na: http://www.code.colostate.edu/my-favorite-rounded-corner-solutions.aspx

[17] (2014) jQuery Corner Demo. Dostopno na: http://malsup.github.com/jquery.corner.js

[18] (2014) CodeMirror editor. Dostopno na: http://codemirror.net/

[19] (2014) JqGrid knižnica. Dostopno na: http://jqgrid.com/

[20] (2014) Navodila za instalacijo JqGrid. Dostopno na: http://www.trirand.com/jqgridwiki/doku.php?id=wiki:how_to_install/

[21] (2014) Rešen problem spreminjanja velikosti stolpca v JqGrid. Dostopno na: http://www.trirand.com/blog/?page_id=393/bugs/column-resize-mark-position/

[22] (2014) Perl definition. Dostopno na:
http://www.techterms.com/definition/perl

[23] (2014) Perl tutorial. Dostopno na:
http://www.tizag.com/perlT/index.php

[24] (2014) HTML::Template package. Dostopno na:
http://search.cpan.org/ samtregar/HTML-Template-2.6